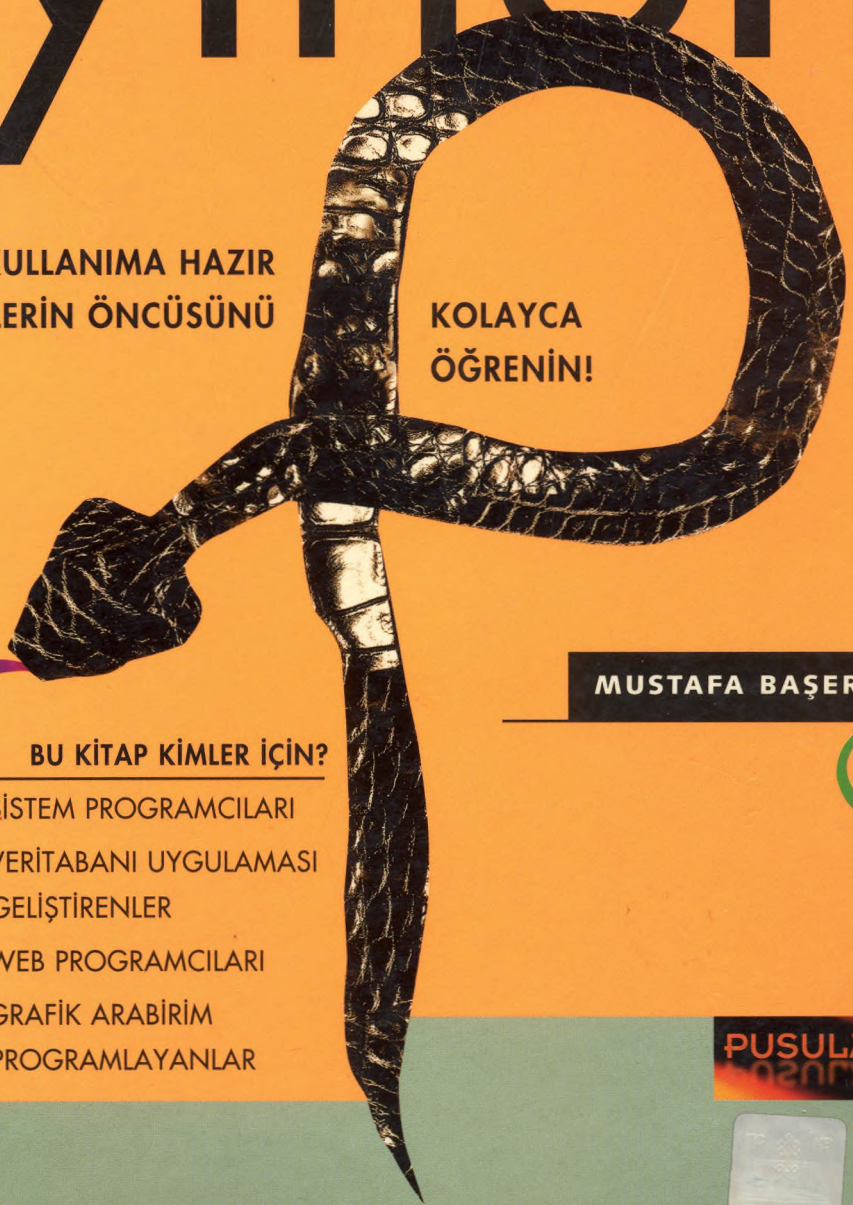
ÇOK AMAÇLI, NESNE TABANLI, MODÜLER PROGRAMLAMA DİLİ

Python



.NET İÇİN KULLANIMA HAZIR DİLLERİN ÖNCÜSÜNÜ KOLAYCA ÖĞRENİN!

MUSTAFA BAŞER

BU KİTAP KİMLER İÇİN?

- SİSTEM PROGRAMCILARI
- VERİTABANI UYGULAMASI GELİŞTİRENLER
- WEB PROGRAMCILARI
- GRAFİK ARABİRİM PROGRAMLAYANLAR

PUSULA

GELECEK Linux Yayınları

Python

Mustafa Başer

PUSULA

YAYINCILIK ve İLETİŞİM LİMİTED ŞİRKETİ

Pusula **87**
Python
Mustafa Başer
1. Baskı: Ekim 2002
Yayın yönetmeni: Mustafa Arslantunalı
Düzelti: Kerem Demirbaş
Sayfa düzeni: Mine Bulut
Kapak illüstrasyonu/tasarım: Mehmet Öztürk

Renk ayrımı film çıkış: Şan Grafik
Tel: (0212) 522 87 79

Baskı: Acar Matbaası
Tel: (0212) 422 18 00
ISBN 975-7092-99-1

Dağıtım sorumlusu: Hüseyin Üstünel
Tel: (0212) 252 42 80

"Kim Korkar..." dizisi, Pusula Ltd. tarafından yayımlanmaktadır. Bütün hakları Pusula Ltd.'e aittir. Yayıncının yazılı izni olmaksızın kitabın tümü veya bir parçası tekrar basılamaz. (Ticarî amaçlar için kullanılmamak şartıyla, fotokopi yoluyla çoğaltılabilir.)

Pusula Yayıncılık ve İletişim Ltd.
Mebusan Yokuşu Akyol Sokak Even Apt. No: 55/3 Cihangir/İstanbul
Tel. : (0212) 252 42 80 pbx
Faks : (0212) 293 15 44
e-posta : pusula@pusula.com
web : www.pusula.com

Mustafa Başer

1970 yılında Yozgat'ta doğdu. ODTÜ Fizik Öğretmenliği bölümünü bitirdi. Yüksek lisans çalışmalarını yine aynı üniversitenin Fen Bilimleri Eğitimi bölümünde yaptı. Fen ve matematik eğitimi dalında doktora çalışmalarına devam etmektedir. Bilişim sektöründe ve çeşitli eğitim kurumlarında çalışmakta, CHIP, PC Magazin, PCNET ve PC World gibi popüler bilgisayar dergilerine Python ve Linux üzerine yazılar yazmaktadır. Ayrıca Linux, Windows NT, Python, Samba gibi konularda ders vermektedir. Linux ve Python ile uğraşmadığı zamanlarda mangal yakmak, kırlarda gezintiye çıkmak, kitap okumak gibi hobiler geliştirmiştir. Eşi Dilek ile Fatih ve Melike isimli iki çocuğu vardır.

ÖNSÖZ

Tamamı telif olan bu kitabın hazırlanmasındaki tek gaye 4 yıllık Python programcılığı sırasında kazanılan bilgi birikimi ve tecrübelerin okuyucu ile paylaşılmasıdır. Konular yazılırken bir programcının ihtiyacı olan bilgiler doğrudan verilmeye çalışıldı. Kitabın kapsamı mümkün olduğu kadar geniş tutulurken, gereksiz bilgilerin içerilmemesine gayret edildi.

Kitap başlıca dört kısım olarak incelenebilir.
1. Python programcılığına giriş (1-13. Bölümler)
2. Veritabanı programcılığı (14-15. Bölümler)
3. GUI programcılığı (16. Bölüm)
4. Web programcılığı (17. Bölüm)

Kitaptaki bölümler sırayla okunmalı ve verilen örnekler iyice incelenmelidir. Birince ve ikinci kısmı bitiren bir programcı kendi alanına göre 3. veya 4. kısmı okuyabilir. Son iki kısım birbiri ile bağlantılı değildir.

Kitap ile ilgili düzelti, açıklama ve kitap içerisinde geçen örneklerin tam kodlarını http://python.kolayweb.com adresinde bulabilirsiniz. Bu adrese, yazarın çeşitli dergilerde Python hakkında yazdığı veya yazabileceği yazılar birer yıllık gecikmelerle konulacaktır.

İçindekiler

01. Python Nedir? **1**

- Python Programlama Dilinin Özellikleri 1
- Nesneye Yönelik Bir Dildir 2
- Özgürdür 2
- Yorumlamalı/Derlemeli 3
- Taşınabilir 3
- Güçlüdür 4
- Hızlıdır 5
- Ticari Uygulamalar Geliştirilebilir 5
- Yazılımı Kolaydır 6
- Öğrenimi Kolaydır 7
- Nerede Kullanılır? 7
- Veritabanı Programcılığı 8
- İnternet Programcılığı 9
- Hızlı Prototip Yazma 10
- GUI Programcılığı 10
- Python Ticari Firmalar Tarafından Kullanılıyor mu? 10

02. Python'a Giriş **13**

- Sisteminizde Python 13
- Etkileşimli Python Kabuğu 14
- Python'da Toplama ve Çıkartma İşlemi 17
- Python'da Çarpma ve Bölme İşlemi 18
- Bir Sayının Kuvvetini Alma 19
- Karmaşık Sayılar 20

03. Python Değişkenleri **23**

- Değişken İsimleri 23
- Sayı Değişkenler 24

• Cümle Değişkenler 25
• Listeler 29
• Tüpler 37
• Sözlükler 38
• Dosyalar 42

04. Cümle İşlemleri 47
• Konumlama ve Parçalama 47
• Cümle Atamaları 48
• Cümle Biçimleme 50
• Büyültme-Küçültme 52
• Desen Bulma 54
• Desenlere Ayırma ve Birleştirme 56
• Desen Değiştirme 57
• Ortalama 57
• Kaçış Karakterleri 59

05. Denetleme Deyimleri 61
• Karşılaştırma Operatörleri 61
• Mantıksal Operatörler 63
• if Deyimi 63
• in Deyimi 65
• if-else 66
• if-elif-else 71
• for Deyimi 72
• break 75
• for-else 76
• continue 77
• while Deyimi 78
• İleri for 79

06. Fonksiyonlar 81
• Fonksiyon Nedir? 81
• Fonksiyonların Tanımlanması ve Çağrılması 83
• Bilgi Girişi ve Dönüşü Olmayan Fonksiyonlar 84

• Bilgi Girişi Olup Geriye Birşey Döndürmeyen Fonksiyonlar 84
• Bilgi Girişi Olmayıp Geriye Bilgi Döndüren Fonksiyonlar 87
• Bir Bilgi Girişi Olup Bilgi Döndürmeyen Fonksiyonlar 92
• Bilgi Girişi ve Dönüşü Olan Fonksiyonlar 96
• Menüler 96
• Fonksiyonlarla İlgili Hazır Fonksiyonlar 100
• map 100
• lambda 102
• İsim Alanları 103
• Fonksiyon Açıklamaları 104

07. Python Modülleri 107
• Neden Modüler Programcılık? 107
• Modüllerin Yazılması ve Çağrılması 108
• Modüllerin Yeniden Çağrılması 113
• Modüllerin Çağırılış Yöntemleri 114
• import <modül> 114
• reload <modül> 115
• from <modül> import <nesne> 115
• from <modül> import * 116
• from <modül> import <nesne> as <yeni_isim> 117
• import <modül> as <yeni_isim> 117
• Hangi Modülde Hangi Nesneler Var? 117
• Özel İsim ve Fonksiyonlar: _isim_ 118
• Program İsimleri: _name_ 118
• Dosya İsimleri: _file_ 121
• Açıklama Satırları: _doc_ 121
• Örnek Modülün Çalıştırılması 123
• Modüllerim Nerede? 124
• Bir Modülü Ne Zaman Çağırmalı? 125
• Tüm Modüllerimi Derlemek İstiyorum 125

08. Hatalarla Başetme 127
• try-except 128
• Hata Uyarılarını Belirleme 130

• Hataları Kullanıcıya Bildirme 131
• try-except-else 134
• Hata Yükseltme 135

09. Düzenli İfadeler 139
• Karakter Eşleme 140
• Meta-Karakterler 143
• Tekrarlamalar 144
• Eşlenen Nesnesinin Özellikleri 145
• Harf Duyarlılığına Son 146
• Re Grupları 146
• Desenlere Ayırma ve Yer Değiştirme 147
• Re'leri Derlemek 149

10. Sınıflar 151
• Sınıfların Oluşturulması 152
• Sınıflara Veri Aktarımı 154
• Sınıf İçi Fonksiyonların Tanımlaması 154
• Banka Uygulaması 156
• Miras Alma 157
• Operatör Yüklemesi 159

11. Dosya, Klasör ve Süreç İşlemleri 161
• os Modülü 161
• os.name 161
• os.environ 162
• os.system(), os.popen(), os.popen2() 163
• os.chmod(), os.chown 166
• os.listdir() 167
• os.link(), os.symlink(), os.unlink() 167
• os.remove(), os.rmdir() 168
• os.mkdir() 168
• os.rename() 168
• os.chdir(), os.getcwd() 168
• os.stat() 168

• os.path 169
• shutil.copytree(), shutil.rmtree() 172
• Dosya Kilitleme 172

12. İnternet Uygulamaları 177
• urllib 177
• ftplib 178
• poplib 180
• smtplib 183
• SimpleHTTPServer 184
• webbrowser 185

13. Çok Kullanılan Standart Modüller 187
• tempfile 187
• random 187
• md5, base64 188
• pickle 189
• ConfigParser 190
• zipfile 192
• crypt, getpass 194

14. Veritabanı-1: shelve 199
• shelve 199
• shelve Uygulaması 202

15. Veritabanı-2: MySQL 205
• MySQL Kurulumu 205
• MySQL Komut Satırı ve İlk Bağlantı 207
• MySQL'de Veritabanı Açılması ve Silinmesi 208
• MySQL'de Kullanıcı Açmak 209
• Tablo İşlemleri 210
• Python-MySQL 217

16. GUI 221
• Pencere, Widget ve Olay 221

• Merhaba GUI 222
• Pencere Başlıkları 225
• Düğmeler ve Geri Çağırımlar 226
• Geri Çağrımlara Veri Aktarımı 230
• Tkinter Olayları 232
• Çoklu Widget'ler 232
• Yazı Tipleri ve Renkleri 235
• Widget Boyutları 237
• Widget Görüntülerini Değiştirmek 239
• Çerçeveler 239
• Durum 240
• Yerleşke 240
• İşaretçi 240
• Üst Düzey Pencereler 240
• Diyaloglar 241
• Bilgilendirme Diyalogları 242
• Dosya Açma Diyaloğu 243
• Entry Widget'i 246
• Message Widget'i 247
• Checkbutton Widget'i 248
• Radiobutton Widget'i 251
• Hepsi Bir Arada 253
• Diğer Geometri Yöneticileri 257
• Grid 257
• Place 259
• Resimler 259
• Listbox ve Kaydırma Çubuğu 260
• Text Widget'i 261
• OptionMenu Widget'i 264
• Menüler 265
• Çizimler 270
• Görsel Araçlar 271

17. Web Programlama 273
• Web Programcılığı Nedir? 273

• Neler Bilmeliyim? 275
• CGI Programlama 276
• Merhaba CGI 278
• CGI Modülü 280
• Saldırılara Dikkat! 285
• Çerezler (Cookie) 286
• Dosya Yükleme 289
• Apache'de Kod Hataları 291
• Linux'ta Diğer Web Programlama Metotları 292
• Linux'ta PSP 293
• IIS'de Python ile ASP 299
• HTMLGen 304

Python Nedir?



Bu bölümde Python programlama dilinin özelliklerini anlatmaya çalışacağız. Ancak ne kadar da anlatmaya çalışsak bu dile yabancı olan birisi için bazı ifadeler anlamsız gelecektir. Çeşitli örnekler vererek zengin bir anlatım hedefliyoruz. Vereceğimiz örnekleri hemen anlamanız elbette ki beklenemez. Eğer daha önceden bir programlama dili bilmiyorsanız, verilen örnekleri anlamaya çalışmamalısınız, nasılsa öyle kabul etmelisiniz. Herhangi bir dil biliyor olsanız bile bazı örnekler önceki bilgilerinizle çelişecek ve hatta size saçma bile gelecektir. Her ne şekilde olursa olsun, bu bölümü "at gözlüklerinizi takarak" okuyunuz. Python'un sihirli dünyası önce beyninizde fırtınalar estirecek, taşları yerli yerine oturtamayacaksınız. Ancak bu fırtına dindikten sonra belki de kitabın geri kalan bölümlerini okumak için geceleri uykusuz bile kalacaksınız. Çünkü bu dil tamamen "insan nasıl düşünüyorsa öyle" yazılmıştır. Geliştiricisi Guido von Russom'un ikinci dili olan Python üst seviyede birisinin kuşbakışı yaklaşımıyla geliştirilmiş, her şey denetim altına alınmış ve dizayn yapılırken insanın düşünme tarzı temel alınmıştır. Programcılıkta birazcık ilerleyince hiç bilmediğiniz şeyleri bile kestirecek "Bunu bilmiyorum ama mutlaka böyledir" diyeceksiniz ve düşündüğünüz şeyin doğruluğunu onu yapınca göreceksiniz. Şunu unutmayınız ki Python'un tamamını öğrenmek neredeyse imkânsız ve aslında gereksizdir de. Bu kitap size sadece rehberlik yapacak, size Python'un olağanüstü dünyasının kapısını aralayacak ve içeriye girdikten sonra sizi kendi halinize bırakacaktır. Bundan sonrası "nasıl olmasını düşünüyorsanız" öyle olacaktır. Bu seviyeye gelince kendinize olan güveniniz artacak ve özgüveniniz doruk noktaya ulaşırken duyduğunuz haz da zirvede olacaktır. Buyurun, bu dilin gizemlerini beraberce aralayalım.

Python Programlama Dilinin Özellikleri

Python'u diğer programlama dillerinden ayıran birçok özellik vardır. Her ne kadar yeni bir dil olmasa bile birçok yönü insanlar tarafından bilinmemek-

tedir. Burada hem bilinmeyenlere biraz ışık tutacak hem de Python'u öğrenmenin artık kaçınılmaz olduğunu göstermeye çalışacağız. Bu mükemmel dilin bazı özellikleri şöyle:

Nesneye Yönelik Bir Dildir

Python'da her şey bir nesnedir. Python programlama dili nesne özellikli dillerin en tepesinde oturmaktadır. Bir zamanlar dünyayı kasıp kavuracağı düşünülen Java'nın bile ötesindedir. Tüm değişkenler, fonksiyonlar, sabitler modüller, sınıflar birer nesnedirler. Şunu hemen ekleyelim ki, sınıf özelliğine sahip dillere de nesne tabanlı diller denmektedir ki Python tüm bu dillerin özelliklerini saklamakla birlikte nesnellik biraz daha ötedir ve daha farklı anlaşılır. Örneğin bir cümle sabiti bile nesnedir. Aşağıdaki örnek bunu en güzel bir şekilde gösteriyor:

```
>>> 'python'.upper()
'PYTHON'
```

Burada 'python' cümle sabiti bir nesnedir ve bu nesneye ait **upper()** fonksiyonu hemen uygulanarak sabit içerisindeki harfleri büyük harflere çeviriyor. Gördüğünüz gibi daha sabit tanımlanır tanımlanmaz bir nesne haline geliyor ve buna çeşitli özellikler bağlanıyor. Elbette ki cümlelerin tek özelliği **upper()** fonksiyonu değildir. İleride cümleler üzerindeki çeşitli özellikleri göreceğiz. Bir örnekte sayı değişkenlerden görelim:

```
>>> a=199
>>> a.__hex__()
'0xc7'
```

Bu örnekte **a** isimli bir tamsayı değişkeni oluşturduk ve bu değişkene **199** değerini atadık. Daha değişkeni tanımlar tanımlamaz bu bir nesne haline gelir ve özellikleri belirir. Bunlardan bir tanesini bu değişkene uygulayarak, tamsayının on altılık (hexadecimal) gösterimini elde ettik.

Özgürdür

Python özgür bir dil olup kaynak kodu herkese açıktır. Kaynağı C ile yazılmıştır. Birçok modülü yine Python iledir, ancak bazı temel modüller ile kendisi tüm zamanların vazgeçilemez ve eskitilemez dili C/C++ ile yazılmıştır. Kaynak kodu üzerinde istediğiniz değişikliği yapabilir ve kendi amaçlarınız

doğrultusunda kullanabilirsiniz ve hatta kendi Python dağıtımınızı bile gerçekleştirebilirsiniz, ancak yazardan izin almak ve yaptığınız değişiklikleri olduğu gibi belirtmek kaydıyla.

Dilin orijinalini istediğiniz medya ortamında dağıtıp, ister ticari isterse özgün/serbest yazılımlar yazabilirsiniz ve bunlar için herhangi bir telif veya dağıtım hakkı için ücret ödemezsiniz. İstediğiniz kadar makinaya kurup her amaç için kullanabilirsiniz, hatta satıp para bile kazanabilirsiniz.

Yorumlamalı/Derlemeli

Python her ne kadar yorumlanan diller sınıfına girse de, aslında derlemeli diller kategorisine de bir ayağını uzatarak tam orta yerde durmaktadır. Yazdığınız kod önce Python yorumlayıcısı tarafından sizin isteğiniz dışında "byte-code" denilen bir biçime derlenir ve daha sonra yorumlanır. Böylece Python her iki dilin özelliğini de taşır. Ancak derleme çok hızlıdır ve ara adımlar gerektirmez. Programın bir sonraki çalıştırılışında kaynak kodda bir değişiklik yoksa yeni bir derleme yapılmadan doğrudan byte-code kullanılır. Hatta kaynak kodu silerek işlemlerin biraz daha hızlanmasını sağlayabilirsiniz.

Bununla birlikte byte-code'un çalıştırılabilmesi için yine de Python yorumlayıcısına ihtiyaç vardır.

Son zamanların popüler ortamı **.net**'e özgü çeşitli Python derleyicileri yazılmıştır. Bu sayede Python yorumlayıcısına gerek kalmadan derlenen programlar **.net** ortamında çalışabilmektedir. **.net**'i UNIX ortamına taşıma gayretleri sürmektedir ve henüz kullanılabilir değildir. Bu gerçekleştiğinde Windows üzerinde derlediğiniz bir Python programı hiçbir değişikliğe gitmeden UNIX ortamında çalışabilecektir ya da tersi.

Taşınabilir

Python platformlar arası taşınabilirdir. Hemen her işlemci ve işletim sisteminde çalışabilir. Bunlar arasında hemen hemen tüm UNIX tipi işletim sistemleri, Windows 9X, 2000, XP, MacOs... sayılabilir.

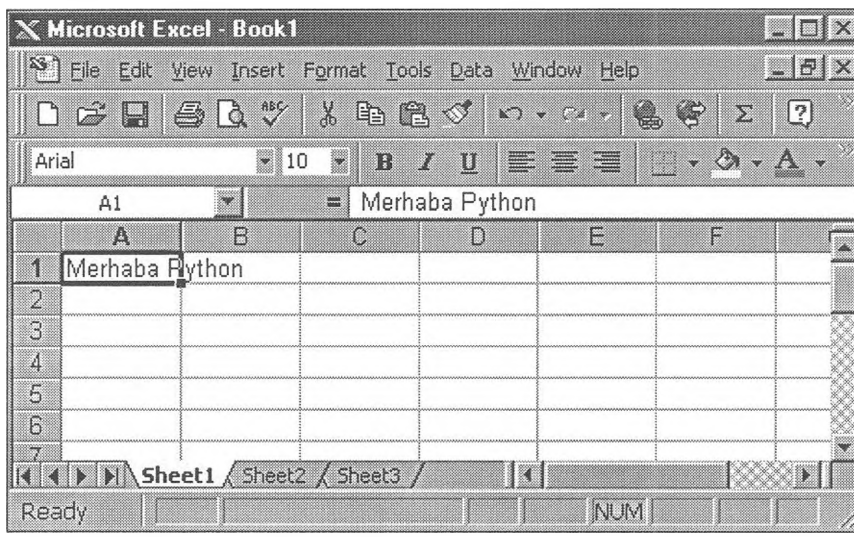
Yazdığınız kaynak kodlar ile byte-code'ların, işletim sistemi ve/veya donanım (işlemci/mimari) bağımlılığı yoktur. İşletim sistemine özgü fonksiyon veya modülleri kullanmadığınız sürece tüm kod ve byte-code'ları işletim sistemleri arasında transfer edip çalıştırabilirsiniz.

Güçlüdür

Python inanılmaz bir güce sahiptir. Aradığınız hemen her şeyi bulabilir, her türlü uygulamayı geliştirebilirsiniz. Python ile yapamayacağınız hemen hemen hiçbir şey yok gibidir. UNIX üzerinde çalışıyorsanız tüm C/C++ dinamik kütüphanelerini çağırabilir ve Python'un rahatlığı altında kullanabilirsiniz. Windows DLL'lerinin Python içerisinden kullanımı ile ilgili çalışmalar henüz sonlanmamıştır, ancak güçlü COM desteği sayesinde Visual C++ ile yapabileceğiniz hemen her şeyi Python ile yapabilirsiniz. COM (Active X) desteği Python'da o kadar ileridir ki, tüm Windows uygulamalarına erişebilir, üzerinde istediğiniz her şeyi yaptırabilirsiniz. Örneğin aşağıdaki kod çalıştırıldığında bir MS-Excel uygulaması başlatılıyor ve birinci satır birinci sütuna, "Merhaba Python" yazdırılıyor:

```
import win32com.client
o = win32com.client.Dispatch("Excel.Application")
o.Visible = 1
o.Workbooks.Add()
o.Cells(1,1).Value = "Merhaba Python"
```

Yukarıdaki kodlarının verdiği pencere **Şekil-1.1**'de görülüyor.



Şekil-1.1: COM desteği ile tüm Windows uygulamaları elinizin altında

Eğer iyi bir C/C++ programcısı iseniz Python nesne ve kütüphanelerini bu dillerin altından çağırıp hayatınızı kolaylaştırabilir ve birçok gereksiz kodu yazmaktan kurtulabilirsiniz. C/C++ API'si oldukça gelişmiş ve kullanımı çok rahattır.

Her programcının Python'un tamamen kendi amaçları için geliştirildiği iddiasında bulunması, bu dilin gücünün bir diğer göstergesidir. Yani bu dilde herkes için bir şeyler mutalaka vardır, hatta en iyisi vardır.

Hızlıdır

Python benzerleri arasında oldukça hızlıdır. Aslında benzerleri yoktur ve bir alt seviyede olan Java ile aralarında gözle görülebilecek bir hız farkı vardır. Java/Python arasındaki karşılaştırmayı Python'un ana sitesinden (http://www.python.org) bulabilirsiniz.

Python'u hızlı kılan etkenlerin başında modüllerin ana sistemden bağımsız olması gelir. İlk başlarda bu size biraz yorucu gelecektir, ancak sizin isteğiniz dışında hiçbir modül hafızaya yüklenmeyecek, sistemi yormayacaktır. Bu sizin tüm programı kontrol altında tutmanızı sağlarken, sisteminizin de hızını artıracaktır. Bazı kritik modüller daha hızlı olsun diye C/C++ ile yazılmıştır.

Güçlü "Garbage-Collector" (çöp toplayıcısı) sayesinde tanımlanan nesnelere olan bağlantılar sonlanınca (artık işe yaramayacağı kesinleşen kodlar) hemen bellekten silinecek, programınız ne kadar büyük olursa olsun, sisteminizin performansı etkilenmeyecek ve hafızanız gereksiz yere kirletilmeyecektir. Böylelikle daha sağlam programlar yazabilecek ve uzun menzilli kullanabileceksiniz. Python ile yazdığınız sunucular çok uzun sürelerde çalışacak ve sistemi yormayacaktır.

Ticari Uygulamalar Geliştirilebilir

Programcıların yorumlamalı dillerden uzak durmalarının bir nedeni de ticari uygulamalar geliştirirken kaynak kodun açık olarak verilecek olması endişesidir. Eğer Python ile ticari bir uygulama geliştirecekseniz böyle bir kaygıya kapılmanıza gerek yoktur. Çünkü kaynak kodu vermek zorunda değilsiniz. Programlarınızı müşterilerinize byte-code ile verebilirsiniz. Ancak tersine mühendislik bu olağanüstü dil üzerinde de etkisini hemen göstererek byte-code'ların bir kısmının orijinaline çok yakın halini elde etmeye yönelik prog-

ramlar ortaya çıkmıştır. Bunu önlemek için; **py2c** ile yazdığınız programı tamamen Python için yazılmış C modülü haline getirebilir ve derleyebilirsiniz. Böylelikle kodunuzun orijinal haline dönüştürülmesini engellemiş olursunuz. Ancak yine de Python yorumlayıcısına ihtiyaç duyulacağını hatırlatalım.

Yazılımı Kolaydır

Kullanımı ve yazılımı en kolay dil olarak dillere destan olmuştur. Bazı programcıların yazılımı üzerinde çok büyük eleştirileri olmuşsa da Python'u öğrendikten sonra yaptıkları eleştirinin çok yersiz olduğunu kendileri ifade etmişlerdir. Hatta birçok C/C++ programcısı, Python'u öğrendikten sonra yazdıkları C kodlarının bile çok temiz ve herkes tarafından anlaşılır olduğunu söylüyor.

Yazılımı oldukça açık olan bu dil sayesinde, yıllar sonra bile koda bakınca hemen ne yaptığınızı anlayacak ve değişiklikler yapabileceksiniz. Sizin yazdığınız kodu başkaları hemen değiştirebilecek, siz de başkalarının yazdığı programları hemen çözecek ve kendi amaçlarınız doğrultusunda düzenlemeler yapabileceksiniz. Oysaki birçok programlama dilinde yazılan kod, yazarı tarafından bile 3 ay sonra anlaşılmaz hale gelmektedir. Şimdi en kolay diller olarak bilinen Pascal ve Perl ile aynı ifadeleri yazalım ve sonunda Python ile karşılaştıralım:

Pascal Kodu:

```
if a=3
begin
        writeln ('Üç');
end;
```

Perl Kodu:

```
if (a==3) {
        print "Üç\n";
}
```

Python Kodu:

```
if a==3:
        print "Üç"
```

Yukarıda gördüğünüz gibi Python hem kolay anlaşılır ve hem de kodları oldukça kısadır. Aynı ifadeler için Pascal'da 28, Perl'de 23 Python'da ise sadece 16 karakter kullandık. Elbette bu kadar kısacık kodda bu fark pek birşey anlatmayabilir. Eğer sayfalar dolusu program yazacaksanız, kod kısalığının farkına varacaksınız. Aradığınız birçok fonksiyonu hazır bulacak, alt seviye programcığı ile uğraşmayacaksınız.

Öğrenmesi Kolaydır

Öğrenmesi en kolay dil olduğunu biz iddia etmiyoruz, bu dili kullanan ve öğrenenler ile, öğrenme üzerine uzmanlaşmış eğitmenler söylüyor. Aslında yazarın kendisi de öğrenim konusunda bir uzman olarak sadece bu iddiayı doğrulamakla yetiniyor. Şimdi vereceğimiz bir örnekle Python öğrenmenin ne kadar kolay olduğunu görelim: "elma" kelimesindeki harfleri sıra ile ekrana bastırmak isteyelim. Yazacağımız kod sadece bu cümle üzerinde bir döngü oluşturmak olacaktır:

```
for i in "elma":
        print i
```

Ne listelerle ne işaretçilerle, ne de ayrımlarla uğraşmanıza gerek yoktur. Değişkenler için tip bildirimi gibi, kullanıcının zihnini karıştıran hiçbir şeyi Python'da bulamazsınız. Değişkenler daha tanımlanırken ne tipte olduğu Python tarafından algılanır ve işlemler yürütülür. Bir program içerisinde aynı değişkeni aynı isim alanında, farklı tiplerde kullanabilirsiniz. Tanım ve ifadelerle ilgili geniş açıklamaları sonraya bırakarak başka bir konuya geçelim.

Nerede Kullanılır?

Python'u istediğiniz hemen her yerde kullanabilirsiniz. Sınırlarınızı aşıp tüm programları bu dil ile yazabilirsiniz. Her programcı kendisi için hazırlanmış birçok kütüphane ve fonksiyonu hazır bulacaktır. Python genel amaçlı bir dil olmakla berber, sanki yazdığınız program için özel olarak gelitrilmiş gibidir. Örneğin Python kendisini "web için tasarlanmış" bir programlama dili olarak ilan etmez. Eğer işiniz web üzerinde uygulama geliştirmek ise Python'un, kendisini bu uğurda adamış birçok programlama dilinden bile üstün olduğunu göreceksiniz. İşiniz dağıtık sistemlerle ise ve sunucu/istemci tipi çalışıyorsanız yine göreceksiniz ki sanki Python dağıtık sistemlerde program geliştiren kodcuların işini kolaylaştırmak için yazılmıştır. Bunun gibi birçok alanda Python'u kullanabilirsiniz. Genel amaçlı bir dil olması, si-

zin alanınıza en uygun dil olmasını engellemiyor. Yapay zekâ programcılığı için özel olarak geliştirilen bir dilin yazarı *"Gördüm ki Python'da benim geliştirdiğim dilin birçok özelliği var, hatta bazı yerlerde benim dilimden bile ilerde"* diye itiraf etmesi sanıyorum konuyu biraz daha anlaşılır yapıyor.
Şimdi kullanabileceğiniz bazı alanları şöyle sıralayalım:

Veritabanı Programcılığı

Eğer karmaşık veritabanı programcılığı ile uğraşıyorsanız, ve dikkatinizi sadece veritabanı ve sorgulamalara yoğunlaştırmak istiyorsanız, programlama dilinin kaprisleri ve zorlukları ile uğraşmak istemiyorsanız Python tam size göre. Hermen hemen her veritabanı sistemi ile barışıktır. Bunlar arasında en yaygın olanlarını şöyle sıralayabiliriz: ODBC, *SQL, gdbm, bdb, xbase...

Bunların yanı sıra, birçok yerde herhangi bir dış veritabanı sistemine ihtiyaç duymayacağınız **shelve** modülü de vardır. Python programcıları çok nadir olarak herhangi bir SQL veritabanı kullanır. Birçok yerde **shelve** ve **pickle** işinizi tamamen görür. Örneğin bir sınıfta öğrencilerin notlarını saklayacağımız bir veritabanı düşünelim. Sınıflar, öğrenciler ve dersler üçgeninde basit bir tasarımı şu şekilde yapabiliriz:

sınıf:öğrenci:ders:not

Bunu şu şekilde bir sözlüğe atabiliriz:

```
dbf['2A']{     'Melike Başer':{'matematik':9,'fen':10},
               'Fatih Başer':{'matematik':10,'fen':10}}
```

Şu anda size çok karmaşık gibi gelen bu yapı aslında oldukça kolaydır. 2A sınıfının öğrencilerini istersek

```
dbf['2A'].keys()
```

Fatih Başer'in fen dersinin notunu istersek:

```
dbf['2A']['fatih başer']['fen']
```

şeklinde bir çağrım yeterli olacaktır. Tüm bu verileri **pickle** veya **shelve** ile diske kaydedip daha sonra üzerinde istediğiniz gibi değişiklik yapabilirsiniz.

Sözlük dediğimiz bu yapı üzerinde SQL tipi sorgulamaların da yapılabildiği çeşitli modüller bağımsız Python programcıları tarafından yazılmıştır. Bunları internetten rahatlıkla bulabilirsiniz. Hatta bazıları o kadar abartmış ki, Python ile SQL sunucusu ve istemcisi bile yazmıştır. Python'un doğal SQL modülü olarak ün yapmış bu modülü yine internetten indirip kullanabilirsiniz. Böylelikle herhangi bir dış veritabanı sistemine ihtiyacınız kalmaz.

Yine de diğer programlar ile de denetlenebilen SQL sunucularına rahatlıkla erişip tüm sorguları kolayca yapabilirsiniz. Python ile en çok kullanılan sunucuların başında PostgreSQL ve MySQL gelmektedir. Hatta dilerseniz MS-Access dosyalarını bile çağırıp istediğiniz sorgulamayı gerçekleştirebilirsiniz.

İnternet Programcılığı

Python'u kullananların birçoğu web programcılığında diğer dillerin zayıflıklarından yakınıp yeni bir arayış içerisinde olanlardan oluşur. Kitabın yazarı da bunlardan biridir. Temel düşüncesi şuydu: "Kullanımı kolay olsun, gereksiz kodlarla uğraşmayayım, Perl ve PHP ile çok zor yaptıklarımı kolayca yapayım ve bunlardan daha hızlı olsun." Sonunda tüm bu kriterleri sağlayan Python'la ve daha sonra bahsedeceğimiz Zope'la tanıştı.

Python'un COM desteğinin mükemmel olduğunu söylemiştik. Bu nedenle IIS4.0 ve üzeri web sunucularında ASP kullanırken VBScript yerine Python kullanılarak hem performans artışı sağlanır hem de ASP programcılığı oldukça kolaylaşır.

Eğer kullanıcıdan alınan verileri disk üzerinde tutuyorsanız, kullanıcının sisteminize dosya yüklemesine izin veriyorsanız, dosya sisteminiz üzerinde kendisine verdiğiniz alanda oynama yapmasına olanak sağlıyorsanız güvenlik ile ilgili problemleriniz var demektir. İnternette "kullanıcıdan gelen bilgiye güvenme" sloganı ne kadar bilinse de birçok sunucunun hacklendiğini her gün duyuyorsunuz. Güvenli internet/web programcılığı istiyorsanız, ilk koşul dil olarak Python kullanmanızdır. Python ile veriler üzerindeki tüm denetimleri istemeseniz bile yapmak zorundasınız, yoksa ikide bir **exception** verecek ve programınız çalışmayacaktır. Dolayısı ile program çok dikkatli yazılmak zorundadır ve her veri sıkı sıkıya denetlenmelidir. Böylelikle güvenlik doruk noktaya çıkar. Konuyla ilgili ayrıntıları kitabın ilgili bölümüne bırakıyoruz.

Hızlı Prototip Yazma

Eğer çok büyük sistemler üzerinde çalışacaksanız, sistemin neye benzediğini görmek için mutlaka bir prototip yazarsınız. Yazacağınız prototip kimi zaman asıl program yerine geçecektir. İşte hızlı bir şekilde uygulama geliştirmek ve sonucunu görmek istiyorsanız, başlayın Python ile kodlamaya.

Birçok protip programı Python ile yazılmış ve fakat performans/güvenirlilik/verimlilik gibi şartlar değerlendirmeye alınıldığında, prototipin asıl program yerine geçtiği görülmektedir.

GUI Programcılığı

Günümüzde artık programlama dilinden kastın kullanıcı grafik arabirimi olduğu şeklinde bir yanlış algılama mevcut. Python size her türlü programlama tekniğinin en iyisini verirken masaüstü kullanıcılarını da unutmaz. Doğal GUI arabirim Tk'dır. Bu toolkit (araç) hem Windows hemde UNIX üzerinde ön tanımlı olarak gelir ve widgetlar bulunduğu platformun doğal görünümü ile aynı olur. Diğer toolkit ile de programlama yapmanız mümkün. Bunlar arasında dünyayı kasıp kavuran Qt (KDE) ve GTK (Gnome) vardır. Her iki araç da hem Windows hem de UNIX üzerinde bulunabilir ve Python ile çok uyumlu çalışırlar. İki dakikada GUI programcılığı nasıl yapılır görelim:

```
from Tkinter import *
Label(None, {'text': "Hemencecik GUI", Pack:{}}).mainloop()
```

Yukarıdaki programın sonucunu **Şekil-1.2**'de görüyorsunuz. Gördüğünüz gibi iki satırda GUI programları yazabilirsiniz.



Şekil-1.2:
Python ile anında GUI.

Python Ticari Firmalar Tarafından Kullanılıyor mu?

Son zamanlarda birçok ticari firma Python'u kendi bünyesinde yaygın bir şekilde kullanmaktadır. Bunların başında Linux'u Linux yapan Red Hat firması geliyor. Hemen her uygulamasını Python ile geliştiriyor. Bunuda daha başta Linux'u sisteminize kurdurturken başlatıyor. Evet yanlış duymadınız, kullandığınız Red Hat Linux'lar Python programları tarafından makinenize

kuruluyor. Python yılanından etkilenip ismini de Anaconda koydular Red Hat kurulum programının. Sadece kurulum programı değil, Red Hat tarafından yazılan ister TUI isterse GUI olsun hep Python kullanılıyor. Oysaki Red Hat bünyesinde çok iyi C/C++ programcıları da vardır, fakat uygulamalar hep Python ile geliştiriliyor.

Microsoft bir ticari uygulamasının bir kısmını Python ile yazdı. **.net** ortamında Python'un yaygınlaşması için büyük çaba sarfediyor. Hatta **C#**'ın Python derleyicisi olarak kullanılmasına imkan tanıyacak çabalar sarfediyor.

Dünyanın en hızlı arama motoru Google birçok işlemde Python'u tercih etti. Google ana arama motoru hariç, son kullanıcıya internet üzerinden verdiği her hizmeti Python ile yazıyor ve sürekli Python bilen elemanlar arıyor.

Python'u yaygın olarak kullanan bir diğer internet portalı da Infoseek'dir. Son kullanıcı arayüzününün tamamına yakınında Python'u tercih ediyor.

Bu rüzgardan Yahoo da nasibini almış olmalı ki bazı servislerinde bu dili kullanıyor.

NASA gizli uygulamalarını Python ile geliştiriyor. Özellikle güvenliğin ön plana çıktığı (Güvenlikten sadace verilerin çalınması ya da sisteme hackerların sızması değil, programın sürekli ve veri kaybına neden olmadan çalışmasını da anlamalısınız.) uygulamalarda bu dili tercih ediyor ve yaygın bir şekilde kullanıyor.

Ünlü içerik yönetim sistemi Zope Python ile yazılmıştır. Python'un yazarı Guido'nun da desteği altında yürütülen bu projenin henüz bir benzeri yazılamadı ve yazılması da imkansız. Ne olduğunu merak ediyorsanız *http://www.zope.org* adresine bir gitmenizi ve incelemenizi tavsiye ediyoruz.

Python ile yazılmış web tarayıcısı grail (*http://grail.python.org*) son zamanların ilgi çeken uygulamalardandır.

Son olarak ülkemizden bir örnek vermek istiyorum; kolaymail.com hemen hemen tüm servislerini Python ile veriyor.

Python'a Giriş

02

Bu bölümde Python kabuğuna kısa bir giriş yaparak hem ilk tanışmamızı gerçekleştireceğiz hem de basit aritmetik işlemleri nasıl yapabileceğimizi göreceğiz. Bu bölümü okurken bir bilgisayar başında olmanızı ve verdiğimiz her örneği uygulayarak sonucunu görmenizi tavsiye ediyoruz. Bu tavsiyemiz sadece bu bölüm değil bundan sonraki tüm bölümler için de geçerlidir.

Sisteminizde Python

Birçok işletim sistemi üzerinde Python kurulu olarak gelir. Eğer sisteminizde yok ise, işletim sisteminize uygun dağıtımı Python'un ana sitesi *http://www.python.org*'dan indirebilirsiniz. Burada her platform için binary ve kaynak kodları mevcuttur ve hiçbir ücret ödemeksizin her amaç için kullanabilirsiniz. Windows kullanıcıları için tavsiyemiz *http://www.activestate.com*'daki Active Python'u kullanmalarıdır. Bu kitabın çoğu yerinde Active Python kullanılmıştır. Active Python, **msi** uzantısı ile dağıtılmakta ve kurabilmek için MSI kurucusunun sisteminize bulunması gerekmektedir. Sisteminizde bulunmuyorsa *www.microsoft.com* adresinden indirip kurabilirsiniz.

Eğer UNIX kullancısı iseniz Python mutlaka sisteminizde kuruludur. Ana programın kurulu olduğu yeri öğrenmek için

```
$ which python
```

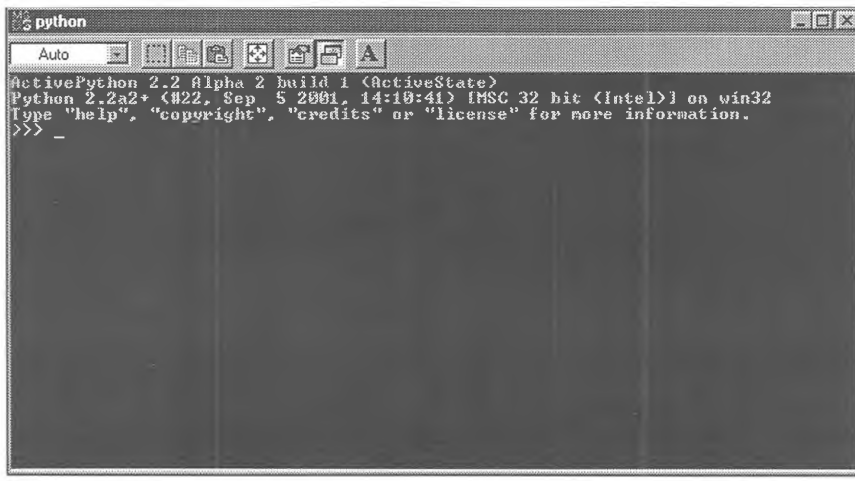
komutunu çalıştırıp, tam patikasını (path) öğrenmeniz gerekiyor, çünkü tüm kodlarınızın başında kullanacaksınız. Eğer bu komut herhangi bir sonuç vermiyorsa sistem yöneticinize danışarak yerini öğrenebilirsiniz. Size kurulu olmadığını söylerse, kurmasını rica edin. Aslında kendi ev dizininize de kurup kullanabilirsiniz. Bunun için Python kaynak kodunu indirerek prefix

değişkenleri ile oynamanız gerekiyor. Bu kitabı okuyorsanız Python'u öğrenmeye karar vermişsinizdir ve eğer kararınızı verdiyseniz, bu işlemleri zaten yapabiliyor olduğunuzu varsayarak kurulumunu sayfalarca anlatmıyoruz.

Etkileşimli Python Kabuğu

Python programlarını iki şekilde yazabilirsiniz. Birincisi etkileşimli olarak, diğeri ise kodları bir dosyaya yazarak, bu dosyanın Python tarafından yorumlanmasını (daha doğrusu önce byte-code'a derlenip sonra bunun yorumlanması) sağlayabilirsiniz. Etkileşimli kullanmak çoğu zaman pek anlam taşımayabilir. Ancak kısa programcıklar yazacaksanız, unuttuğunuz bir fonksiyonun nasıl kullanıldığını denemek istiyorsanız, hemen Python kabuğunu açıp istediklerinizi yapabilirsiniz. Bu kısa bölümde Python'un etkileşimli kabuğunun nasıl kullanıldığını temel artimetik işlemler ile birlikte göreceksiniz.

Windows kullanıcıları için üç kabuk mevcut, bunlardan birincisi **Şekil-2.1**'de görülen standart kabuktur. Bu kabuğa Başlat/Programlar/ActiveState ActivePython x.x/Python Interpreter Shell yolunu kullanarak erişebilirsiniz. Bu kabuk Windows kullanıcıları için sevimsiz, UN*X kullanıcıları içinse tanıdık gelecektir. Kullanımı gerçekten zor olan Sandart Interpreter Shell'i kullanmanızı tavsiye etmiyoruz.

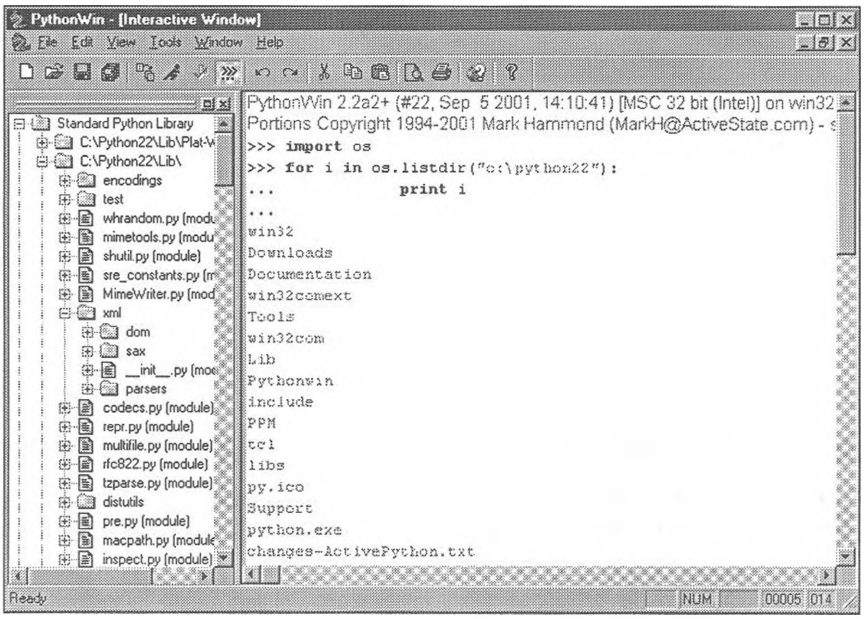


Şekil-2.1: Standart Python kabuğu. Siyah ekranda sevimsiz UNIX tipi bir kabuk

Her kabukta standart olan, kabuk imlecinin ">>>" şeklinde olmasıdır. Bu imleç Python'un komut ya da ifadeleri girmek için hazır olduğunu gösterir. Eğer yok ise, başka bir şeyler girilmesini bekliyor demektir.

Bir diğer kabuk ise kolay kullanımlı ve renkli olan PythonWin Environment'dir. Bunun örneğini de **Şekil-2.2**'de görüyorsunuz. Kitabımızda bu kabuğu kullanacağız. Active Python kurmuşsanız, erişmek için Başlat / Programlar/ActiveState ActivePython x.x/PythonWin Environment patikasını takip ediniz.

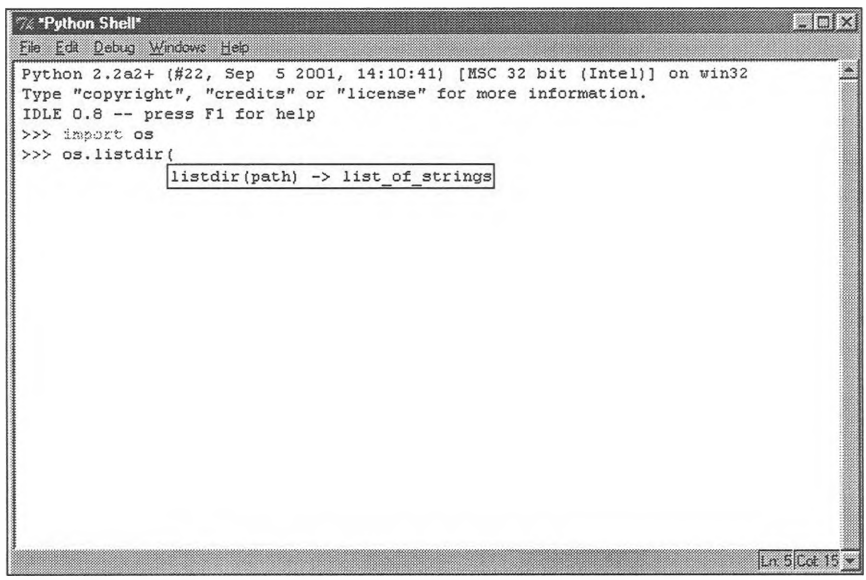
Bu kabukta önceki yazdığınız ifade ve komutlara yukarı ok tuşuna basarak ulaşabilirsiniz. Bulunduğunuz satır bloğunun tamamını yeniden düzenlemek ve çalıştırmak için enter tuşuna basmanız yeterli olacaktır.



Şekil-2.2: Windows kullanıcıları için alternatif Python kabuğu PythonWin

PythonWin sadece bir kabuk değil aynı zamanda bir geliştirme ortamıdır, çünkü Python'a özel bir metin düzenleyicisi, sınıf tarayıcısı ve debugger'i ile birlikte program yazmayı adeta bir zevk haline getiriyor.

Son olarak bahsedeceğimiz ve kitabımızdaki ekran görüntülerinin çoğunu aldığımız kabuk ise Python'unun geliştiricisi Guido tarafından yine Python ile yazılan Idle denilen Bütünleşik Geliştirme Ortamıdır. Idle sadece bir kabuk değil aynı zamanda birçok aracı içeren bir yazılımdır. Windows üzerinde yazılım geliştirecekseniz içerisinde bulunan metin düzenleyicisi sayesinde Python kodlarınız rengarenk olup anlaşılırlığı artıyor. Bu kadarla da bitmiyor Idle, programı yazarken kullanacağınız modül ve fonksiyonlarla ilgili online yardımları da hemen ekranda belirtiyor. Daha siz fonksiyon yazılımını bitirmeden açtığınız ilk parantezde alabileceği parametre ve tipler ile ilgili bir açıklama kutucuğu çıkıyor alt tarafta. **Şekil-2.3**'te örnek bir Idle penceresi görüyorsunuz:



Şekil-2.3: Idle ile Python programları yazmak hem zevkli hem de oldukça kolay. Windows ve Unix'te çalışabiliyor.

Hem UNIX hem de Windows platformlarında çalışan Idle'ı Windows'da Python'un kurulu olduğu klasör altındaki tools\idle\idle.py ile çalıştırabilirsiniz.

UNIX kullanıcıları için ise iki kabuğu önerebiliriz. Birisi standart Python kabuğu ki en çok kullanılanıdır. Bunu çalıştırmak için

```
$ python
```

komutunu kullanmanız yeterli olacaktır. Bir diğeri ise X-Windows altında çalıştırabileceğiniz Idle'dır. Idle Python ile standart olarak gelmez, sonradan kurmanız gerekebilir. Bazı Linux dağıtımlarında ön tanımlı olarak kurulu olabilir. Eğer zevkli Python programları yazmak istiyorsanız siz de mutlaka Idle kullanmalısınız.

Şimdi Python komut satırında temel işlemleri yaparak programlama dili ile tanışalım.

Python'da Toplama ve Çıkartma İşlemi

Python'da toplama işlemini klasik artı işareti ile gerçekleştirebilirsiniz: "+"

2 ve 3 sayılarını toplamak için:

```
>>> 2+3
5
```

Görüldüğü gibi matematiksel işlemleri tanıdık bir şekilde yapıyoruz. Python yaptığınız her işlemden sonra, sonucu anında ekrana yazacaktır. Sonuçlar gösterilirken ">>>" şeklindeki imleç olmayacaktır.

Basit bir toplama gibi görünen bu işlemde aslında yapılan şey, iki tamsayı sabit nesnesi üzerinde "+" operatörünün işletilmesi ile elde edilen sonucun ekrana basılmasıdır. Python da operatörler nesneler üzerine etki eder ve sonucunda yine bir nesne ortaya çıkartır. Bazı operatörler birden fazla nesne tipi üzerine etki edebilirler. Örneğin toplama operatörü ilerde göreceğimiz gibi cümle, liste ve tüp nesneleri üzerinde işlem yapabilir. Hatta bazı operatörler farklı tipteki iki nesne üzerine bile etkiyebilirler, basit bir örnek vermek gerekirse tamsayı nesnesi ile reel sayı nesnesinin toplaması olabilir. Şimdi böyle bir örnek görelim:

```
>>> 3+7.2
10.199999999999999
```

Toplama operatörünün etkilediği ilk nesne 3 tamsayısıdır, ikincisi ise 7.2 reel sayıdır. Sonuç ise tahmin edeceğiniz gibi bir reel sayıdır. Yukarıdaki işlemin sonucu biraz mantıksız gibi gelebilir. Python'un neden böyle davrandığını ilerki konularımızda açıklayacağız. Şimdilik işlemlerinizi yaparken elde

edeceğiniz sonucun nasıl olacağını kestirmeniz ve program akışını buna göre yazmanız yeterli olacaktır.

İki sayının farkı alınırken ise "-" operatörü kullanılacaktır. 100 ve 72 arasındaki farkı bulalım:

```
>>> 100-72
28
```

Daha değişik örnekleri kendinizi deneyerek sonucunu tahmin ediniz ve uygulayarak tahmininiz ile sonucu karşılaştırınız. İkisi farklı ise sebebini düşününüz.

Python'da Çarpma ve Bölme İşlemi

Çarpma işlemi için "*" kullanıyoruz. Çarpma işlemi cümleler, listeler ve tüpler için de geçerlidir. Basit bir çarpma işlemi yapalım:

```
>>> 45*98
4410
```

Şimdi bir bölme işlemi yapalım:

```
>>> 100/25
4
```

Şu örneğe bakalım:

```
>>> 9/5
1
```

Neden böyle oldu? Python aritmetik işlemleri yaparken, kullanılan sayıların hangi cinsten olduklarını iyi bilir. İki tamsayı ile işlem yapıldığında sonuç daima tamsayı olarak verilir, ondalık kısım atılır. Şimdi de **9/5.0** işlemini yapın (Evet, evet ondalık kısmı da yazıyorsunuz)

```
>>> 9/5.0
1.8
```

Şimdi istediğiniz gibi bir sonuca ulaştınız sanırım. Çarpma ve çıkartma işlemleri de benzer şekilde yapılmaktadır. Peki birleştirilmiş işlemlerde durum nedir? Genel kural Python'da da geçerli. Öncelik sırası şöyle: Çarpma, bölme, toplama ve çıkartma. Birden fazla işlemi bir anda yaparken parantezler kullanarak gruplandırma yapabilirsiniz. Bu durumda önce parantezin içi yapılır ve işlemler soldan sağa doğru ilerler. Şu iki işlemin sonucunun neden farklı olduğunu düşününüz:

```
>>> 9.0/5*3
5.4
>>> 9.0/(5*3)
0.6
```

Bir Sayının Kuvvetini Alma

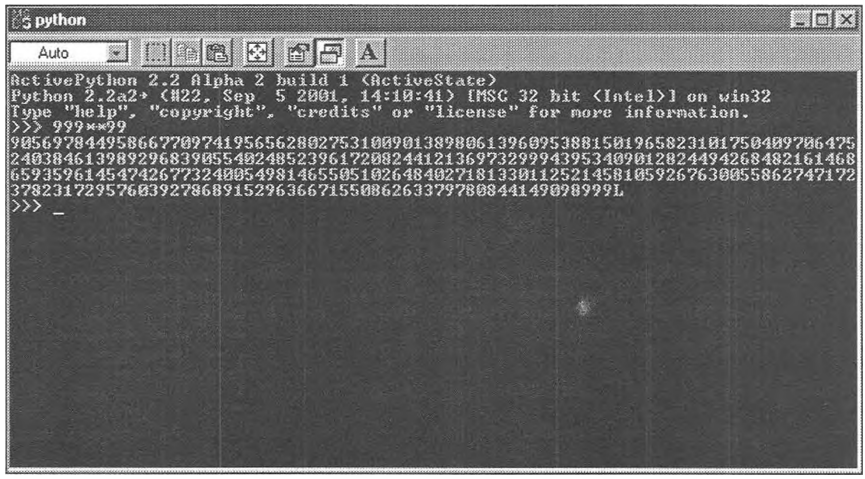
Pythonda bir sayının kuvvetini almak istiyorsanız, çift yıldız "**" operatörünü kullanabilirsiniz. Örneğin 8^2 işlemi için:

```
>>8**2
64
```

Python oldukça büyük ve küçük sayıları kullanabilir ve üzerlerinde işlemler yapabilir. Bunun doğal sonucu olarak nümerik matematik dalında araştırma yapan uzmanların gözde programlama dili olmuştur, işte size basit bir örnek:

```
>>> 999**99
905697844958667709741956562802753100901389806139609538815019658 2
310175040970647524038461398929683905540248523961720824412136973 2
999439534090128244942684821614686593596145474267732400549814655 0
510264840271813301125214581059267630055862747172378231729576039 2
786891529636671550862633797808441490989991
```

Sondaki L işareti uzun bir sayı olduğunu gösteriyor. Bu işlemin sonucunu **Şekil-2.4**'de görüyorsunuz.



Şekil-2.4: Python çok büyük sayılarla çalışırken size hiçbir zorluk çıkarmaz.

Karmaşık Sayılar

Python sadece tam ve ondalık sayılarla işlem yapmaz. Onun karmaşık sayılarla da arası çok iyidir. Bu tip sayılarda karmaşık kısım **j** ile belirtilir. Hemen iki karmaşık sayıyı toplayalım (Aslında aşağıdaki parantezleri kullanmanıza gerek yok, biz iki sayıyı ayırmak için koyduk.)

```
>>> (2+4j) + (5-1j)
(7+3j)
```

Aynı iki sayı arasında çarpma işlemini yapacak olursak:

```
>>> (2+4j) * (5-1j)
(14+18j)
```

Bölme işlemi de geçerli:

```
>>> (2+4j) / (5-1j)
(0.230769230769+0.8461538461541j)
```

Şimdi diyeceksiniz ki; "Karmaşık sayılarda ondalık kısım bulunmadığı halde iki bölmenin sonucunda ondalık kısımlar vardır." Evet doğru, karmaşık sayılarda bölme işleminde bu kural gözetilmez (adı üstünde karmaşık sayı).

Normal programcılar için karmaşık sayılar ile işlem yapmak pek anlamlı olmayabilir, ancak matemetik alanında araştırma yapanlar için bu olağanüstü bir özellik olarak algılanır.

Python'da kullanabileceğiniz birçok hazır matematik fonksiyonu vardır, bunları ilerde göreceğiz.

Python Değişkenleri 03

Python standart olarak aşağıdaki hazır değişkenler ile birlikte gelir. Bunlarla her işinizi yapabildiğiniz gibi, kendinize ait değişken türleri de tanımlayabilirsiniz.

- sayılar
- cümleler
- listeler
- tüpler
- sözlükler
- dosyalar

Değişken İsimleri

Değişken isimlerini seçerken dikkat edeceğiniz iki nokta var:

Rakam ile başlayamazlar.

Harf dışı karakter barındıramazlar (Türkçe harf, parantez, ünlem gibi. Fakat Linux'ta Python2.1'den itibaren Türkçe harflerden oluşan değişken isimleri tanımlayabilirsiniz.)

Değişkenler tanımlanırken ne türden olduğunu belirtmenize gerek yoktur. Yani başına "$" , "%" , "@" gibi tuhaf şeyleri yazmak, hangi değişken için ne kullanmanız gerektiğini hatırlamak gibi bir kaygınız yok. Aynı değişkeni bir seferinde sayı, diğer seferinde sözlük olarak kullanabilirsiniz.

Python'da aksi belirtilmedikçe tüm değişkenler yerel olarak algılanırlar. İsim alanı ile ilgili açıklamalarımızı yeri geldiğince açıklayacağız.

Sayı Değişkenler

Sayı değişkenler üzerinde tüm aritmetik işlemleri yapabilirsiniz, sonucu daha sonra kullanmak üzere sayı değişkenlere atayabilirsiniz. Atama işlemi "=" operatörü ile yapılır. Hemen örnek verelim:

```
>>> a=5
>>> b=7
>>> c=a+b
```

Gördüğünüz gibi ekrana yazılmadı. Çünkü sonucu bir değişkene atadık. Herhangi bir atama işlemi yapılmıyorsa etkileşimli ekranda sonuç standart çıktıya (burada ekrana) yazdırılır. İmleçte **c** yazıp enter tuşuna basarsanız, **c** değişkeninin değerini görürsünüz.

```
>>> c
12
```

Şimdi hemen şu işlemi yapmaya çalışalım: **c+d**

```
>>> c+d
Traceback (innermost last):
  File "<stdin>" , line 1, in ?
NameError: d
```

Ne bekliyordunuz? Sonucun **12** olmasını mı? Python'u ilk öğrenen herkes bu hataya sürekli düşer. Python'da bir değişken bir işlem içerisinde kullanılmadan önce mutlaka tanımlanmalıdır. Ön tanımlı değerler "boş" ya da "sıfır" olarak alınmazlar (diğer betik programlama dillerinin aksine). Burada verilen hatanın analamını ve hata yakalamayı başka bir bölümde inceleyeceğiz. Şimdilik bu uyarının bir hatadan ibaret olduğunu bilmeniz yeterli. Bu işinizi biraz zorlaştırıyormuş gibi gelebilir, ama aslında ilerde doğabilecek birçok problemi önceden halletmenize yarar. Belki tanımlanmamış sayı değerlerin ön tanımlı değerinin sıfır olup olmaması problem oluşturmayacaktır, ancak diğer değişkenleri kullanmaya başlayınca işler değişiyor.

Şimdi de şöyle bir işlem yapmaya çalışalım:

```
>>> a=9
>>> a=a+8
```

İyi de bu nasıl oluyor? Yani nasıl oluyorda bir sayıya birşeyler ekleyipte yine kendisini elde ediyoruz? Eğer herhangi bir programlama dili bilmiyorsanız bu soruyu mutlaka kendinize sormuş olmalısınız. Buradaki eşitlik matematikte olduğu gibi her iki tarafın birbirine eşit olduğunu göstermiyor, sağ taraftaki işlemin sonucunun sol taraftaki değişkene atanacağı anlamına geliyor. Yani **a** değişkeninin değerine **8** ekleyip sonucu yeniden **a** değişkenine atıyoruz. **a** değişkeninin son değerine bakalım:

```
>>> a
17
```

Karmaşık sayıları da değişkenlere atayabilirsiniz. Örneğin:

```
>>> a=5+3j
>>> b=17-9j
>>> a+b
(22-6j)
>>> a*b
(112+6j)
```

Cümle Değişkenler

Bunlar çoğumuz için tanıdıktır. Ancak yine Python'a özgü şeylerin olduğunu da baştan belirtelim. Örnek:

```
>>> a="Bu ne lahana, bu ne turşu?"
>>> a
'Bu ne lahana, bu ne tur\376u?'
```

Bu da ne? Yoksa Python cümle değişkenlere Türkçe harfleri atarken problem mi çıkartıyor? Hayır, merak etmeyin, sorun yok. Burada yazılan sadece bu değişkenin Python tarafından nasıl algılandığı, ekrana yazdırılırken farklı olacaktır. İsterseniz deneyelim:

```
>>> print a
Bu ne lahana, bu ne turşu?
```

Bu arada **print** hazır fonksiyonunun ne işe yaradığını da gördük. Bir değişkeni ya da önüne yazılan sabiti ekrana yazdırıyor. Aslında ekrana yazılan bir nesnenin cümle cinsinden ifadesidir. İlerde değişik nesleri de ekrana yazdıracağız.

Python'da cümle sabitlerini değişkenlere atarken iki yolumuz var. Bunlardan birisi çift tırnak işareti (") ile başlayıp bitirmek, diğeri ise kesme işareti ile ('). Bu ikisi arasında Python için pek bir fark yok. Ancak kullanım açısından farklar ortaya çıkabilir (Merak etmeyin Perl'deki gibi anlamsız bir iş yaparak, cümleler içerisinde değişkenleri yazmayacağız). Örneğin; sabit içerisinde bir kesme kullanacaksak, burada (") kullanmak en mantıklısı:

```
>>> il="Ankara'dan mı geliyorsunuz?"
>>> print il
Ankara'dan mı geliyorsunuz?
```

Cümlemizde tırnak işareti kullanacaksak bu sefer tersi:

```
>>> renk='"Kırmızı rengi çok severim."  dedi.'
>>> print renk
"Kırmızı rengi çok severim."  dedi.
```

Eğer her seferinde standart bir atama kullanmak istiyorsanız o zaman kaçış karakterlerini kullanmak zorundasınız (Kaçış karakterleri ilerde daha detaylı anlatılacak). Örneğin:

```
>>> satir='İzmir\'in kavakları.'
>>> print satir
İzmir'in kavakları.
>>> ses="Kedi \"miyav\" dedi."
>>> print ses
Kedi "miyav"  dedi.
```

Şimdi şöyle bir işlem yapmaya çalışalım:

```
>>> a=9
>>> b="5"
>>> c=a+b
Traceback (innermost last):
  File "<stdin>" , line 1, in ?
TypeError: number coercion failed
```

Hatanın sebebini çıkartabildiniz mi? Eğer bir Perl kullanıcısı iseniz bunun size çok saçma gelmesi normal. Ancak Python kendi içerisinde çok tutarlıdır ve sizin hata ile de olsa yanlış işlemler yapmanıza izin vermez (Çünkü bunlar ilerde daha büyük belaları doğurabiliyor). Python'da cümle ve sayı değişkenler birbirlerinden tamamen ayrıdırlar. Aralarında işlemler yapılırken mutlaka biri diğerine çevrilmelidir. Yukarıdaki **a+b** işleminin iki anlamı olabilir. Birisi normal toplama yapılacak, diğeri ise 9 ve 5 sayıları arka arkaya yazılacak (Sonucun 95 olmadığını bir kez daha hatırlatalım). Aynen şöyle:

```
>>> c=a+int(b)
>>> print c
14
>>> c=str(a)+b
>>> print c
95
```

int() ve **str()** fonkiyonlarının açıklamasını az sonra vereceğiz. Burada toplama işleminin cümle ve listeler için de geçerli olduğunu belirtelim. Cümlelerde "+" işlemi, iki cümlenin arka arkaya bitiştirileceği anlamına geliyor:

```
>>> a="Hatasız kul"
>>> b=" olmaz."
>>> a+b
>>> c=a+b
>>> print c
Hatasız kul olmaz.
```

int() fonksiyonu, cümle değerlerini tamsayılara çevirmek için kullanılır. Eğer cümle içerisinde nokta varsa hata mesajı verir

```
>>> int("345")
345
>>> int("123.6")
Traceback (innermost last):
  File "<stdin>", line 1, in ?
ValueError: invalid literal for int(): 123.6
```

Eğer, sayı tamsayı değil de ondalık bir sayı ise o zaman **int()** yerine **float()** fonksiyonunu kullanmalısınız. **int()** fonksiyonuna benzer şekilde **float()** da cümle değerleri ondalık sayılara çevirmek için kullanılır:

```
>>> float("3"); float("9.8")
3.0
9.8
```

Burada noktalı virgülün (;) ne işe yaradığını tahmin ettiniz herhalde: Tek satırda birden fazla komutun çalıştırılmasını gerçekleştiriyor.

str() fonkiyonu ise ondalık ve tamsayıları cümle değerlere çevirir:

```
>>> "1234"+str(567)
'1234567'
```

Eğer Pyhon2.0'dan daha ileri bir sürüm kullanıyorsanız, bir sayı değişkenin değerini cümleye çevirmek için nesne modelini kullanmak daha mantıklıdır. Bunu şu şekilde yapabilirsiniz:

```
>>> a=99
>>> a.__str__()
'99'
```

Görünüşte biraz işi uzattık gibi, ancak nesne tabanlı bir dilde mümükün olduğu kadar nesne özellikleri ile işlem yapıp programımızı daha akıcı ve hızlı yapabiliriz. Sizden sonra program üzerinde değişiklik yapmak isteyenlerde bu isteklerini daha rahat yapabilirler.

Bir cümlenin uzunluğunu **len()** hazır fonksiyonu ile bulabilirsiniz:

```
>>> len('Hatasız kul olmaz.')
18
```

Nesne modeli ile şu şekilde yapabilirsiniz (> Python2.0):

```
>>> 'Hatasız kul olmaz.'__len__()
18
```

Nesne modelini sabitler için kullanmak pek anlamlı olmayabilir ancak değişkenlerde tercih etmelisiniz.

Cümlelerle ilgili daha fazla bilgiyi liste ve tüp değişkenlerini öğrendikten sonra, başka bir bölüm halinde, daha detaylı bir şekilde öğreneceğiz.

Listeler

Bir liste, birden çok cümle veya sayı sabitini (aslını söylemek gerekirse herhangi bir Python nesnesini: fonksiyon, sınıf ya da modül gibi) belirli bir sırada barındıran değişkenlerden ya da sabitlerden oluşur. Liste içerisindeki sabit ya da değişkenlere, bulunduğu konuma göre kolayca erişebilirsiniz. Buna indeksleme diyoruz. Örneğin 'Python', 'Perl', 'Php' sabitlerini ayrı ayrı cümle değişkenlerinde saklamak yerine hepsini bir liste değişkeninde barındırabiliriz. Bunu şu şekilde yapıyoruz:

```
>>> programlar=['Python', 'Perl', 'Php']
>>> programlalar
['Python', 'Perl', 'Php']
```

Listeleri oluşturuken köşeli parantez [ ] kullanırız. Parantezler içerisine, sabitleri (ya da değişkenleri) aralarına virgül koyarak sıralarız. Programlar listesinde üç tane nesne var. Bu nesnelere erişmek için bulundukları yerin indeksini yazarız. İndeksler 0'dan başlar. Erişmek için *liste[indeks]* şeklinde bir yazılım kullanırız. Burada indeks mutlaka bir tamsayı olmak zorundadır. Bu tamsayı herhangi bir değişken tarafından ifade edilebilir.

Şimdi bu listenin ikinci elemanını isteyelim:

```
>>> programlar[1]
'Perl'
```

Benzer şekilde birinci ve üçüncü elemanları istersek:

```
>>> programlar[0]; programlar[2]
'Python'
'Php'
```

Bir listeye daha sonradan eleman ekleyebiliriz. Bunun için iki yol var, ya toplama işlemini kullanmak ya da liste nesnesinin **append()** metodunu kullanmak. Önce toplama ile 'Pascal' elamanını ekleyelim:

```
>>> programlar=programlar + ['Pascal']
>>> programlar
['Python', 'Perl', 'Php', 'Pascal']
```

Bu işlem ile aslında iki listeyi birbirine eklemiş oluyoruz, yoksa "programlar + 'Pascal'" gibi bir işlem yapamayız. Bu elma ile armutun toplanması hikâyesine benzer; yani ayrı cinsten iki nesne birbiri ile toplanamaz (birbirlerine otomatik çevrilebilen nesne tipleri hariç).

Listelerin elemanlarını sondan da alabilirsiniz. "-1" konumu son elemanı barındırır. "-2" ise sondan bir öncekidir.

```
>>> programlar[-1]
 'Pascal'
>>> programlar[-3]
'Perl'
```

Bir listenin istediğiniz iki konumu arasındaki tüm elemanları çağırabilirsiniz. Bunun için *liste[ilk_konum:son_konum]* şeklinde bir yazılım yeterli olacaktır.

```
>>> programlar[1:3]
['Perl', 'Php']
```

Eğer ilk konum belirtilmezse sıfır (0), son konum belirtilmezse ya da liste uzunluğundan daha büyük bir sayı verilirse listenin son elemanına kadar alınır:

```
>>> programlar[:2]
['Python', 'Perl']
>>> programlar[2:]
['Php', 'Pascal']
>>> programlar[2:191]
['Php', 'Pascal']
```

append()

Bir listeye eleman eklemek için toplama işlemi tercih edilmez, yerine liste nesnesinin özelliklerinden **append()** kullanılır. Python'da her şey birer nesne olarak algılanır. Dolayısı ile **programlar** listesi de bir nesnedir. Nesnelerin metotlarını şu yazılımla kullanabilirsiniz:

nesne.metot

Bazı nesnelerin metotları da başka metodlardan oluşabilir (örneğin, **os.path.isdir()**).

Şimdi **programlar** liste nesnesine 'Python' elemanını ekleyelim:

```
>>> programlar.append('Python')
>>> programlar
['Python', 'Perl', 'Php', 'Pascal','Python']
```

Peki liste nesnelerinin başka özellikleri var mıdır? Elbette, birçok işlem yapabilirsiniz. Eğer bir nesne ile kullanılabilecek metotların neler olduğunu görmek istiyorsanız *dir(nesne)* fonksiyonunu kullanabilirsiniz. Buradaki **dir**'in bir klasörün içeriğini ekrana basan komutla bir ilgisi olmadığını hemen belirtelim. Şimdi bir listenin metotlarına bakalım:

```
>>> dir([])
['__add__', '__class__', '__contains__', '__delattr__', '__deli-
tem__', '__eq__', '__ge__', '__getattr__', '__getitem__',
'__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__repr__', '__rmul__', '__setattr__', '__setitem__',
'__setslice__', '__str__', 'append', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
```

Peki [] ne anlama geliyor? Tahmin ettiğiniz gibi boş bir liste anlamında. Burada [] yerine herhangi bir liste değişkeni kullanabilirdik. Programlar listesi için de aynı sonucu elde edebiliriz.

```
>>> dir(programlar)
 ['__add__', '__class__', '__contains__', '__delattr__', '__deli-
tem__', '__eq__', '__ge__', '__getattr__', '__getitem__',
'__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__',
'__init__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__repr__', '__rmul__', '__setattr__', '__setitem__',
'__setslice__', '__str__', 'append', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
```



Bu metotlar Python sürümlerinde farklı olabilir. Ancak her zaman geriye uyumluluk vardır.

Şimdi sırasıyla bu özelliklerin çok kullanılanlarını inceleyelim:

count():

Bir listedeki bir elemanın kaç kere tekrarlandığını verir. Örneğin listemizde 'Python' elemanından iki tane vardır. Bunu bulmak için:

```
>>> programlar.count('Python')
2
```

Eğer istediğiniz eleman yok ise 0 (sıfır) sonucu verilir:

```
>>> programlar.count('Java')
0
```

extend():
İki listenin toplanması için kullanılabilir. Aslında bu metodun yerine normal toplama işlemi de kullanabilirsiniz. Programlar listesine **['Awk', 'Java']** eklemesini yapalım:

```
>>> programlar.extend(['Awk', 'Java'])
>>> programlar
['Python', 'Perl', 'Php', 'Pascal', 'Python', 'Awk', 'Java']
```

Aynı işlemi

```
>>> programlar + ['Awk', 'Java']
>>> programlar
['Python', 'Perl', 'Php', 'Pascal', 'Python', 'Awk', 'Java']
```

şeklinde de yapabilirdik.

index():
İstenilen bir elemanın konumunu (indeksini) verir. Eğer eleman birden fazla tekrarlanıyorsa ilk bulunulan yeri verir.

```
>>> programlar.index('Python')
0
>>> programlar.index('Pascal')
3
```

Liste içerisinde olmayan bir elemanın indexine bakmak istediğinizde hata ile karşılaşırsınız:

```
>>> programlar.index('C++')
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in ?
```

```
    programlar.index('C++')
ValueError: list.index(x): x not in list
```

insert():
Bir listeye istenilen bir konumda ekleme yapmak için kullanılır. Örneğin 'Perl' ve 'Php' arasında 'Wish' oturtalım:

```
>>> programlar.insert(3,'Wish')
>>> programlar
['Python', 'Perl', 'Php', 'Wish', 'Pascal',
'Python','Awk','Java']
```

programlar.insert(3, 'Wish')'in anlamı; programlar listesinin 3. elamanı **'Wish'** olsun ve sonrakilerin konumu bir yükseltilsin.

pop():
Listenin son elemanını çıkartır:

```
>>> programlar.pop()
'Java'
>>> programlar
['Python', 'Perl', 'Php', 'Wish', 'Pascal', 'Python', 'Awk']
```

remove():
Herhangi bir elemanı listeden çıkartmak için kullanabilirsiniz:

```
>>> programlar.remove('Wish')
>>> programlar
['Python', 'Perl', 'Php', 'Pascal', 'Python', 'Awk']
```

Eğer çıkartmak istediğiniz eleman birden fazla kere varsa, ilk olan çıkartılır. Bulunmayan bir elemanı çıkartmaya çalıştığınızda hata ile karşılaşırsınız:

```
>>> programlar.remove('c++')
Traceback (most recent call last):
  File "<interactive input>", line 1, in ?
ValueError: list.remove(x): x not in list
```

reverse():

Bazı durumlarda bir listenin elemanlarını sondan başa doğru sıralamak isteyebilirsiniz. Bunu ilerde göreceğimiz döngülerle yapabilirsiniz, ancak zahmete gerek yok, bunun için **reverse()** metodu sizi bekliyor:

```
>>> programlar.reverse()
>>> programlar
['Awk', 'Python', 'Pascal', 'Php', 'Perl', 'Python']
```

sort():

En çok işinize yarayacak olan metodun **sort()** olduğu kuşkusuz. Liste elemanlarını sıralamak için kullanılabilir. Önce sayı sabitleri sonra cümle sabitleri sıralanır:

```
>>> programlar.append(3)
>>> programlar.append(1)
>>> programlar
['Awk', 'Python', 'Pascal', 'Php', 'Perl', 'Python', 3, 1]
>>> programlar.sort()
>>> programlar
[1, 3, 'Awk', 'Pascal', 'Perl', 'Php', 'Python', 'Python']
```

len()

Bir listenin kaç elemandan oluştuğunu **len()** fonksiyonunu kullanarak bulabilirsiniz:

```
>>> len(programlar)
4
>>> len(['elma','armut'])
2
```

Python2.0'den daha yüksek bir sürüm kullanıyorsanız, şu şekilde de yazabilirsiniz:

```
>>> programlar.__len__()
4
>>> ['elma','armut'].__len__()
2
```

Bir listeyi bir tamsayı ile çarpabilirsiniz:

```
>>> a=[34,'Daha neler']
>>> a*4
[34, 'Daha neler', 34, 'Daha neler', 34, 'Daha neler', 34, 'Daha neler']
```

Şimdi biraz kafanızı karıştıralım:

```
>>> a=[3,7,9]
>>> b=a
>>> b.append(109)
>>> a
[3, 7, 9, 109]
```

Klasik bir programlama mantığı ile bu durumu açıklamak imkansızdır ve sanki Python'da bir bug bulmuş gibi sevinebiliriz. Ancak işin iç yüzü öyle değil. Şöyle bir çözümleme getirelim:

Bir **a** listesi tanımladım ve elemanları olarak da **3,7,9**'u seçtim. Daha sonra **a** listesini **b**'ye atadım. Ben **b** listesine bir eleman ekledim, ancak görüyorumki aynı ekleme **a** listesine de yapılmış.

Aslında tam senaryo böyle değil. Biz bir **a** nesnesi oluşturduk, daha sonra bu **a** nesnesini **b** ismi ile de çağırmayı istedik. Eğer UN*X biliyorsanız bu işlemi bir hard link olarak niteleyebilirsiniz. "**b=a**" ifadesinde bir atama yapılmıyor, sadece **a**'yı **b** ilede çağırabileceğimizi Python yorumlayıcısına bildiriyoruz. Python'a yeni başlayanların en çok düştükleri hatalardan birisi budur. Liste ve sözlükleri başka bir değişkene atamak istiyorsanız, onun bir kopyasını almalısınız. Yani aslında yapmak istediğimiz şey şudur:

```
>>> import copy
>>> a=[3,7,9]
>>> b=copy.deepcopy(a)
>>> b.append(199)
>>> a
[3, 7, 9]
>>> b
[3, 7, 9, 199]
```

Yukarıdaki örnekte "**import copy**" satırının açıklamasını sonraya bırakıyoruz. **copy.deepcopy()** fonksiyonu kenidisine verilen nesnenin bir kopyasını çıkartır.

Tüpler

Tüpler de listeler gibidir. Ancak bir tüpün içeriğini değiştiremezsiniz. Elemanlar parantez içerisinde sıralanırlar. Örneğin:

```
>>> meyvalar=('Elma', 'Armut', 'Portakal')
>>> meyvalar[0]
'Elma'
>>> meyvalar[-2]
'Armut'
>>> meyvalar[1:]
('Armut', 'Portakal')
```

"Peki listeler varken neden tüplere ihtiyacımız var?" sorusunu sormuş olabilirsiniz. İlk nedeni tüplerin içeriğinin sabit kalmasıdır. Eğer yanlışlıkla da olsa değerinin değiştirilmesini istemediğiniz bir değişkene ihtiyacınız varsa o zaman tüpleri kullanmak en iyisi olacaktır. Bir diğer neden ise hafıza yönetimini en aza indirgediğinden program akışını hızlandırmasıdır. Dolayısıyla, program boyunca kullanılacak dizi değerler mutlaka tüplere atanmalıdır.

Tüpleri değişkenlere değer atarken de kullanabilirsiniz:

```
>>> (a,b,c)=(100,'Rakam',6.7)
>>> a
100
>>> b
'Rakam'
>>> c
6.7
>>> a*c
670.0
```

Aslında yukardakinin yerine:

```
>>> a,b,c=100,'Rakam',6.7
```

yazılımını kullanmak da doğrudur.

Sözlükler

Python programlamada sözlükler çok kullanılır. Özellikle veritabanı uygulamalarında sözlükler büyük kolaylık sağlamaktadır. Sözlükler de listeler gibi birden çok elemanı taşırlar. Ancak elemanın konumu sıfırdan başlayan sayılar yerine, sizin belirlediğiniz herhangi bir cümle olabilir. Kullanıcı adları ve bunlara ait parolaların saklanacağı bir sözlük düşünürsek, parolalara kullanıcı adlarına göre erişilmelidir. Yani ben öyle bir değişken tanımlamalıyım ki, ona kullanıcı adını verdiğimde bana parolasını versin. Bunun için en ideal ortam Python sözlüğüdür. Bunu hemen hayata geçirelim:

```
>>> kullanicilar={    'mbaser':'birnisan',
                      'gorkem':'delidana',
                      'akgun':'gizlibirsifre'}
>>> kullanicilar['gorkem']
'delidana'
```

Yani,

sozluk={anahtar1:değer1, anahtar2:değer2}

Şeklinde bir yazılımı vardır. Bir sözlüğün toplam eleman sayısını yine **len()** fonksiyonu yardımı ile bulabiliriz:

```
>>> len(kullanicilar)
3
```

Ya da >Python2.0 için;

```
>>> kullanicilar.__len__()
3
```

Bir sözlükte var olan bir anahtarın değerini değiştirmek isteyebilirsiniz. Bu durumda yazılım şöyledir:

sozluk[anahtar]=degeri
Örnekleyecek olursak;

```
>>> kullanicilar['fatih']='subardagi'
>>> kullanicilar['akgun']='degisik sifre'
>>> kullanicilar
```

```
{'fatih': 'subardagi', 'mbaser': 'birnisan', 'gorkem': 'delida-
na', 'akgun': 'degisik sifre'}
```

keys():
Sözlüklerde anahtarları bilmek kuşkusuz çok önemlidir. Bir sözlüğün anahtarlarını **keys()** özelliğini kullanarak bulmanız mümkün.

```
>>> kullanicilar.keys()
['gorkem', 'mbaser', 'akgun']
```

Gördüğünüz gibi kullanicilar sözlüğünün anahtarlarını bir liste içerisinde aldık.

has_key():
Peki bir sözlükte herhangi bir anahtarın olup olmadığını nasıl bulacağız? Bunu bulmanın en kolay yolu **has_key()** özelliğini kullanmaktır. Eğer aradığınız anahtar varsa sonuç 1, yoksa sonuç 0 olacaktır:

```
>>> kullanicilar.has_key('mbaser')
1
>>> kullanicilar.has_key('dilek')
0
```

values():
Bazen sözlüklerin sadece değerlerini kullanmak isteyebilirsiniz, o zaman **values()** metodu işinize yarayacaktır:

```
>>> kullanicilar.values()
['delidana', 'birnisan', 'gizlibirsifre']
```

items():
Bir sözlüğün içerisindeki tüm (*anahtar, değer*) tüplerini bu özellik yardımıyla elde edebilirsiniz. Bu size şimdilik anlamsız gelse de ileri programcılıkta işleri oldukça kolaylaştırıyor. Özellike web programcılığında sık sık başvurulan bir metottur.

```
>>> kullanicilar.items()
[('gorkem', 'delidana'), ('mbaser', 'birnisan'), ('akgun',
'gizlibirsifre')]
```

Şimdi biraz kafanız karışmış olabilir. Liste içerisinde tüpler var! Nasıl oluyor da oluyor? Daha önceden belirttiğimiz gibi, bir liste veya sözlük içerisindeki elemanlar herhangi bir Python nesnesi olabilir. Yani listenin herhangi bir elemanı;

- başka bir liste,
- bir sözlük,
- bir dosya,
- bir fonksiyon,
- bir sınıf

olabilir. Örneğin bir sözlükteki herhangi bir anahtara ait değer başka bir sözlük olabilir:

```
>>> yiyecekler={
'meyveler':{'turunçgiller':['portakal', 'mandalina'],
                        'erikler':['can eriği', 'malta eriği']},
...                'sebzeler':('ıspanak', 'lahana', 'fasulye')}
```

Birçok yeni Python kullanıcısının bunları gördüğünde panik içerisinde kafasının karıştığı olmuştur. Kullandıkça ne kadar kolay olduğunu göreceksiniz. Burada bir sözlüğümüz var, adı yiyecekler. Yiyecekler iki gruba ayrılmış meyveler ve sebzeler. meyveler de yine iki alt gruba ayrılmış: portakal ve mandalinadan oluşan turunçgiller, can eriği ve malta eriğinden oluşan erikler. Sebzeler ise bir ayrıma tâbi tutulmadan direk bir tüpün içerisine yazılmış.

Şimdi de, sözlüklerin ne kadar yararlı olduğunu görelim:

```
>>> yiyecekler['meyveler']['turuncgiller']
['portakal', 'mandalina']
```

Sanıyorum bu sizlere bir şeyler ifade ediyor. Python'da istediğiniz bir değere erişmenin ne kadar kolay olduğunu bu kısacık örnek gözler önüne seriyor.

get()

Sözlüklerde çok kullanılan metotlardan birisi de **get()**'dir. Bir sözlükte var olan bir anahtarın değerini getirir, eğer anahtar yoksa hiçbir şey getirmez.

"İyi de ben zaten bir anahtarın değerini alabiliyorum." diyebilirsiniz. Şimdi aşağıdaki örneğe bakalım:

```
>>> yiyecekler['bakliyat']
Traceback (innermost last):
  File "<stdin>", line 1, in ?
KeyError: içecekler
>>> yiyecekler.get('bakliyat')
>>>
>>> yiyecekler.get('sebzeler')
('\375spanak', 'lahana', 'fasulye')
```

Burada gördüğünüz gibi yiyecekler sözlüğünde bakliyat diye bir anahtar yok. Eğer siz bunu isterseniz, anahtar olmadığından hata ile karşılaşırsınız (key error hatası). Eğer siz 'bakliyat'ı **get()** ile isterseniz, anahtar olmadığından hiçbir şey dönmeyecektir. **get()**'in ilginç bir özelliği de şudur:

get(x,y)

Eğer bahsi geçen sözlükte *x* anahtarı varsa o gelecek yoksa *y* değişkeni gelecektir:

```
>>> ikaz="Aradığınız anahtar yok."
>>> print yiyecekler.get('bakliyatlar', ikaz)
Aradığınız anahtar yok.
```

İlginç değilmi?

del(), clear()

Bazen bir anahtarı ve değerini sözlükten çıkarmak isteyebilirsiniz. İşte o zaman **del()** fonksiyonunu kullanmalısınız:

```
>>> del yiyecekler['sebzeler']
>>> yiyecekler
{'meyveler': {'turuncgiller': ['portakal', 'mandalina'], 'erikler': ['can eri\360i', 'malta eri\360i']}}
```

Ya da bir sözlükteki tüm anahtarları silmek isteyebilirsiniz. Bunu yapan metot da **clear()**'dır.

```
>>> yiyecekler.clear()
>>> yiyecekler
{}
```

update()
Bir sözlükte bulunan anahtarların değerini başka bir sözlükte bulunan anahtarların değeri ile değiştirmek için kullanılır. Aşağıdaki örneği inceleyelim:

```
>>> sozluk1={'a':1, 'b':2, 'c':3}
>>> sozluk2={'a':5, 'b':5}
>>> sozluk1.update(sozluk2)
>>> sozluk1
{'b': 5, 'c': 3, 'a': 5}
```

Yazılımı şöyle belirtirsek:

a.update(b)

Anlamını da şu şekilde açıklayabiliriz: B'deki bir anahtar a'da varsa, değerini a'daki anahtar da güncelle.

Dosyalar

Bu bölümde Python'da inceleyeceğimiz son veri tipi dosyalardır. Bir dosya açmak için hazır **open()** fonksiyonunu kullanırız. Yazılımı şöyledir:

dosyadegiskeni=open(dosyapatikası, acma_modu)

Bir dosya dört değişik modda açılabilir: Okuma (**r**), yazma (**w**), ekleme (**a**) okuma ve yazma (**rw**). Açacağınız dosya binary (ikili) ise açma modları (**b**) ile bitiştirilmelidir. Eğer açma modu verilmemişse ön tanımlı olarak okunabilir olarak açılacaktır. Yazma modunda dosya açarsanız ve eğer o isimde eski bir dosya var ise bu dosyanın içeriği kaybolacaktır ve size herhangi bir uyarıda bulunulmayacaktır. Olmayan bir dosyayı ekleme modunda açmaya çalışırsanız, o isimde yeni bir dosya oluşturulur.

Dosya patikasını iki şekilde belirtebilirsiniz, bunlardan ilki UNIX tipi belirtim diğeri ise DOS tipidir. Eğer DOS tipi belirtim kullanırsanız, "\" karakteri, ka-

çış karakteri olarak algılanacağından, bunları çift kullanmalısınız. Örneğin **C** diskinin kök dizininde **benim.txt** isimli bir dosya oluşturmak istersek

UNIX belirtimi: **c:/benim.txt**
DOS belirtimi: **c:\\benim.txt**

Ancak yazdığınız program çeşitli platformlarda çalışacaksa en iyisi UN*X tipi belirtimdir ve mümkün olduğu kadar hangi diskte işlem yapılacağı belirtilmemelidir. Eğer disk belirtilmezse dosya programın çalıştırıldığı diskte açılır. Bizim örneğimizde, eğer biz **C** diskinde çalışıyorsak, aynı dosyayı **/benim.txt** şeklinde de belirtebilirdik. Böylelikle sistemler arası tam uyumlu bir patika tanımlamış oluruz.

Bir dosyaya ancak cümle değişkenler yazabilirsiniz. Dosyalarda çok kullanılan metotlar şöyle:

Özellik	**Yaptığı iş**
write(x)	x cümlesini dosyaya yazar.
readline()	Dosyadan bir satır okur.
readlines()	Dosyanın tamamını bir listeye okur.
read(x)	Dosyadan x byte okur. Sonraki okumalara son bulunan yerden başlar.
read()	Dosyanın tamamını bir cümle değişkene okur.
close()	Dosyayı kapatır.

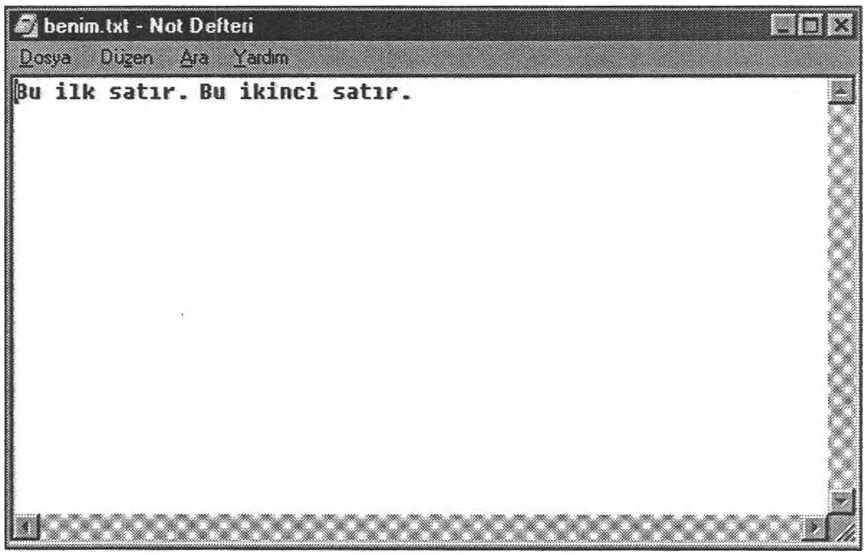
Dosyalarla ilgili diğer özellikleri Python Kütüphane Referansı'ndan öğrenebilirsiniz.

Açmış olduğunuz dosyayı kapatmanıza çoğunlukla gerek yoktur. Program sonlandığında (veya çalışan blok sonlandığında) veya dosya değişkeni başka bir amaçla tekrar kullanıldığında dosya otomatik olarak kapatılacaktır. Ancak program bazen istem dışı sonlanabilir. Bu durumda **write()** ile dosya değişkenine yazdıklarınızın gerçekten dosyaya yazılıp yazılmadığından emin olamazsınız. En iyisi dosya ile işiniz biter bitmez hemen onu kapatmaktır. Gerçek yazma işleminin tamamı ancak dosya kapatıldığında oluşur.

Şimdi yazılabilir bir dosya açalım:

```
>>> ilkdosyam=open('/benim.txt','w')
>>> ilkdosyam.write('Bu ilk satır.')
>>> ilkdosyam.write('Bu ikinci satır.')
>>> ilkdosyam.close()
```

Bu dosyanın içeriği **Şekil-3.1**'de görünüyor.



Şekil-3.1: Python ile oluşturduğumuz ilk dosyanın içeriği not defteri ile görüntüleniyor

Oluşturulan ilk dosya başarısız mı?

Burada yanlış olan bir şeyler var. Biz ikinci satır da görüntülenmesi için "Bu ikinci satır." yazdık ama hepsi ilk satırda görünüyor. Bunun sebebini anlayabildiniz mi? Hemen söyleyelim, satır sonlarına o satırın bittiğini belirten satır sonu karakterini koymadık. Python'da bir satırın bittiği **\n** kaçış karakteri ile belirtilir. Diğer kaçış karakterlerini bu bölümün sonunda bulabilirsiniz. Şimdi dosyayı yeniden şu şekilde oluşturalım:

```
>>> ilkdosyam=open('/benim.txt','w')
>>> ilkdosyam.write('Bu ilk satır.\n')
```

```
>>> ilkdosyam.write('Bu ikinci satır.\n')
>>> ilkdosyam.close()
```

Şimdi dosyamızı doğru şekilde yazmış olduk. Dikkat ettiyseniz dosyamızın içindeki eski bilgiler yok oldu. Eğer varolan bir dosyayı tekrar "**w**" modu ile açarsanız, dosyanın içeriğini kaybetmiş olursunuz. Buna çok dikkat etmelisiniz. Varolan dosyanın arkasına birşeyler eklemek istiyorsanız "**a**" modu ile açmalısınız.

Eğer dosyanız çok büyükse **read()** ya da **readlines()** metotlarını kullanmak pek akıllıca olmasa gerek. Çünkü bu iki metot dosyanın tamamını okur ve hafızaya alır. Programınız çok kullanıcılı bir sistemde çalışıyorsa (CGI'lar gibi) ve bir anda birden çok kullanıcı aynı program ile büyük dosyayları açıyorsa sistem performansını düşürürsünüz. Bu durumlarda en iyisi **readline()** metodunu kullanmaktır. Böylelikle her seferinde dosyadan bir satır okumuş olursunuz. Böyle durumlarda genellikle **while** döngüsü kullanılır. **while** deyimi ile ilgili açıklamaları önümüzdeki bölümlerde göreceksiniz. Şimdilik bu döngüyü kullanarak bir dosyayının şu şekilde okunabileceğini bilmekte fayda var:

```
>>> satir=1
>>> kurulumdosyasi=open('c:/setuplog.txt')
>>> while satir:
        satir=kurulumdosyasi.readline()
        print satir
```

Python2.2'den daha yüksek bir sürüm kullanıyorsunuz **for** döngüsü, açılan dosya üzerinde iterasyon olarak çalışıyor. Yukarıdaki işlem yerine aşağıdaki kodu da kullanabilirsiniz.

```
>>> kurulumdosyasi=open('c:/setuplog.txt')
>>> for s in kurulumdosyasi:
        print s
```

Bir dosyayı okuyup hemen ekrana bastırmak istiyorsanız o zaman dosya değişkeni ve okuma işlemleriyle uğraşmak yerine hepsini tek satırda yapabilirsiniz. Eğer web programcılığı yapacaksanız, bu işlem çok işinize yarayacaktır:

```
>>> print open('c:/setuplog.txt').read()
```

Ya da bir dosya açıp hemen bir şeyler yazmak istiyorsanız, şu şekilde bir yazılım da geçerlidir:

```
>>> open("/deneme.txt","w").write("Bu bir deneme dosyasıdır.")
```

Burada "**w**" modu yerine "**a**" modunu kullanarak var olan bir dosyanın sonuna yazma işlemi gerçekleştirebilirsiniz.

Bir listenin içeriğini olduğu gibi bir dosyaya yazabilirsiniz. Bunun için **writelines()** metodu kullanılmalıdır:

```
>>> a=['Python da ', 'dosya ', 'işlemleri ', 'çok ', 'kolay.']
>>> open("/deneme2.txt","w").writelines(a)
```

Cümle İşlemleri



Günümüz programlarının birçoğu iki eksen etrafında dönmektedir. Metin işleme ve veritabanı uygulamaları. Hiçbir dil yoktur ki bu iki konuda ilerlemiş olmasın. Özellikle Python metin işleme konusunda oldukça ileri düzeydedir. Birçok hazır fonksiyon ve kütüphane ile istediklerinizi kolayca ve zahmetsizce yaparken, diğer dillerde program yazanların çektikleri eziyetlere bakıp gülümseyeceksiniz. Bu bölümde cümlelerle ilgili detaylı bilgileri vereceğiz.

Konumlama ve Parçalama

Cümle sabitlerini aynen birer liste gibi düşünebilirsiniz. Her cümle, elemanları birer karakterden oluşan bir liste gibi davranır. Ancak listelerdeki gibi, herhangi bir karakteri silme, bir yere başka bir karakteri ekleme işlemi yapamazsınız. Bunun için başka yollar izlemelisiniz.

Bir cümlenin istenen bir konumdaki karekterini almayı biz "konumlama"olarak adlandıracağız. Şimdi "Fişini de al Mustafa Bey" cümlesinin bazı karakterlerini konumlayalım:

```
>>> x="Fişini de al Mustafa Bey."
>>> x[1]
'i'
>>> x[15]
's'
```

Cümlelerde indeksler sıfırdan başlar (her zaman olduğu gibi). Sondan başa doğru da indeksleme yapabilirsiniz, bunun için eksi işaretini kullanmalısınız:

```
>>> x[-7]
'a'
```

Peki bu ne demek? Sondan başa doğru 1'den başlanarak istediğiniz karaktere kadar ilerlenecek (Dikkat edin, sıfırdan başlamıyor).

Aralardan parçalar da alabilirsiniz:

```
>>> x[13:20]
'Mustafa'
```

Burada 13 ve 20 sayıları konumlama işleminin nereden başlayıp nerede biteceğini göteriyor.

Kesme işleminde başlangıç belirtilmezse sıfır (0), son belirtilmezse cümlenin en sonuna kadar alınır. Son, cümle sonundan daha büyükse, yine cümle sonuna kadar alınır.

```
>>> print x[:5]
Fişin
>>> print x[9:]
 al Mustafa Bey
>>> print x[7:1555]
de al Mustafa Bey
```

Bir cümlenin son harfini iptal etmek için (Çoğu zaman satır sonu işaretini kaldırmak için kullanılır.) x[:-1] şeklinde kullanabilirsiniz.

Cümle Atamaları

Bazı zamanlarda bir cümleye birden fazla satır atamak isteyebilirsiniz. Bunu yapabileceğiniz iki yol var.

Birincisi satır sonu kaçış karakterini (\n) kullanmak:

```
>>> cumle="Dağ başını duman almış\nGümüş dere durmaz akar"
>>> print cumle
Dağ başını duman almış
Gümüş dere durmaz akar
```

İkincisi ise Python'a ait üç tırnak yazılımını kullanmak (Özellikle uzun metinleri atamak için):

```
>>> cumle="""Dağ başını duman almış
Gümüş dere durmaz akar
Güneş ufuktan şimdi doğar"""
>>> print cumle
Dağ başını duman almış
Gümüş dere durmaz akar
Güneş ufuktan şimdi doğar
```

Bu yazılımı print fonksiyonu ile birlikte de kullanabilirsiniz:

```
>>> print """Dam üstünde saksağan
Vur beline kazmayı"""
Dam üstünde saksağan
Vur beline kazmayı
```

Eğer atama yapacağınız cümle bir satıra sığmıyorsa o zaman satırın devam ettiğini belirten (\) kaçış karakterini kullanmak durumundasınız. Aslında Python satırlarının uzunluğunu kısıtlayacak herhangi bir şey yoktur, ancak programın okunurluğu açısından satırları bölmek daha doğru bir yaklaşım olur:

```
>>> a="Bu cümlenin tamamı bir satır \
... olacak, ancak yazarken iki satıra böldük."
>>> print a
Bu cümlenin tamamı bir satır olacak ancak yazarken iki satıra
böldük.
```

Aslında bu yazılım sadece cümle atamaları için geçerli değildir. Herhangi bir Python ifadesini istediğiniz kadar satıra anlamlı bir şekilde bölebilirsiniz:

```
>>> a=3 +\
... 7 + \
... 9
>>> a
19
```

Cümle Biçimleme

Çoğu programlarda metin açıklamaları yapılırken aralarına bir değişkenin değerini yazmak gerekir. Bunu yapmanın iki yolu var.

İlki print fonksiyonunu parçalı bir şekilde yazmak:

```
>>> a=24
>>> print "Bu müessese ", a, "saat açıktır."
Bu müessese 24 saat açıktır.
```

Görüldüğü gibi bir **print** fonksiyonuyla birden çok cümleyi yazdırmak için aralarına virgül işareti koyuyoruz. **print** fonksiyonu her satırın sonuna satır sonu işareti koyacaktır. Satır sonu işaretinin koyulmasını istemiyorsanız, fonksiyonu virgül "," ile sonladırabilirsiniz.

İkinci ve tercih etmenizde fayda olan yol ise biçimleme kodlarını kullanmak:

```
>>> kahvaltı='7:30'
>>> print "Her sabah saat %s da kahvaltımı yaparım."% (kahvaltı)
Her sabah saat 7:30 da kahvaltımı yaparım.
```

Bu yazılım C/C++ bilenlere tanıdık gelecektir. Biçimleme kodları cümle değişkenlere atama yapmakta da kullanılırlar:

```
>>> yas="7"
>>> oglum="Fatih %s yaşındadır."% (yas)
>>> print oglum
Fatih 7 yaşındadır.
```

Ya da:

```
>>> kizim="Melike %s aylıktır."% ("onyedi")
>>> print kizim
Melike üç aylıktır.
```

Diğer çok kullanılan biçimleme kodları şöyle:

Biçimleme Kodu	Değişken Türü
%s	cümle
%d	Tamsayı
%f	Reel Sayı
%c	Karakter

Şimdi birkaç örneğe bakalım:

```
>>> print "Alan kaydı için %f $ ödedim."% (11.95)
Alan kaydı için 11.950000 $ ödedim.
```

Burada istenmeyen bir durumla karşı karşıyayız. Fazladan sıfırlar eklendi. Bunu engellemek için %f biçimleme kodunda ondalık kısmın uzunluğunu belirtmemiz gerekir:

```
>>> print "Alan kaydı için %0.2f $ ödedim."% (11.95)
Alan kaydı için 11.95 $ ödedim.
```

Bu örnekte de gördüğünüz gibi, ondalık kısmın ilk iki hanesi basılıyor. Şu örnek daha açıklayıcı olacaktır:

```
>>> print "Pi sayısının değeri %0.4f dir"% (3.1415926535897931)
Pi sayısının değeri 3.1416 dir
```

Karakter biçimleme kodu da aynen cümle biçimleme kodu gibi kullanılır:

```
>>> print "Fatih 1%c sınıfında okuyor."% ('A')
Fatih 1A sınıfında okuyor.
```

Biçimleme kodları, anlamlı Python atamaları içerisinde dilediğiniz şekilde kullanılabilir. Aşağıdaki örneklerde sırasıyla bir liste ve sözlük içerisinde kullanılmıştır:

```
>>> a="aksam"
>>> ogunler=["sabah", "ogle", "%s" % a]
>>> ogunler
['sabah', 'ogle', 'aksam']
>>> S={"7.30": "sabah",
...     "12.30": "ogle",
...     "18.30": "%s" % a
...     }
>>> S
{'12.30': 'ogle', '7.30': 'sabah', '18.30': 'aksam'}
```

Cümle atamalarında birden çok ve farklı biçimleme karakteri aynı anda kullanılabilir. Bunun için biçimleme değerleri parantez içinde yazılır:

```
>>> aksamYemegi="Bu gün %s yemeğini saat %0.2f'de yiyeceğiz." %
("akşam", 19.00)
>>> print aksamYemegi
Bu gün akşam yemeğini saat 19.00'de yiyeceğiz.
```

Okunurluğu artırmak için, değerler farklı satırlara yazılabilir:

```
>>> print "Yarın %s dersinden %s sınav, saat %d'de de %s olacağız"
% (
...                                                    "biyoloji",
...                                                    "yazılı",
...                                                    16.30,
...                                                    "sözlü"
...                                                    )
Yarın biyoloji dersinden yazılı sınav, saat 16'de de sözlü olacağız
```

Büyültme-Küçültme

Kimi zaman bir cümlenin tamamını büyük harfe çevirmeye ihtiyacımız olabilir. Bu durumda **upper()** özelliğini kullanmalıyız:

```
>>> a="ekmek arası köfte"
>>> print a.upper()
EKMEK ARASı KöFTE
```

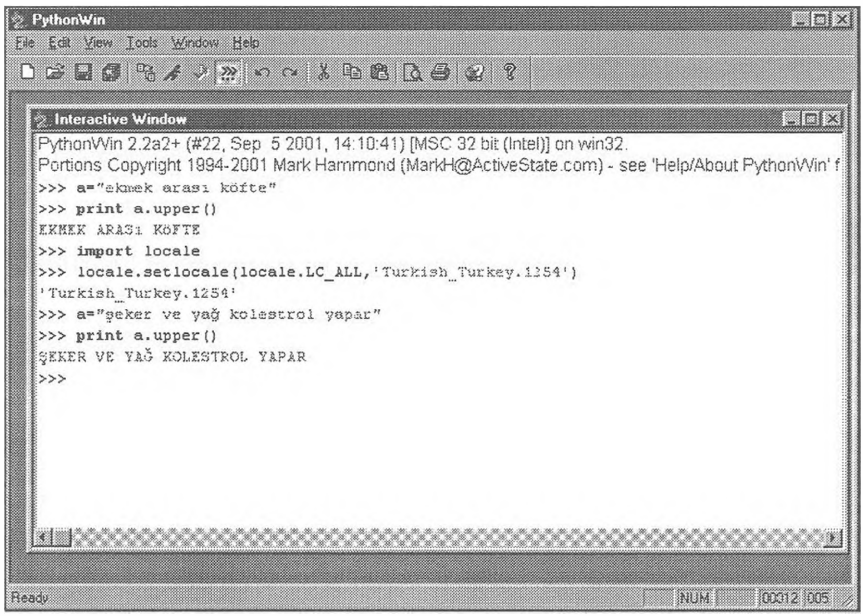
Burada Türkçe harflerin büyültülmediği dikkatinizi çekmiş olmalıdır. Cümleler üzerinde işlemler yaparken dikkat etmeniz gereken bir diğer nokta da **locale**'dir; yani yerellik. Python uluslararası dillere son derece uygun bir yapıdadır. Yazdığınız herhangi bir program içerisindeki metinler de dahil olmak üzere yerelleştirilebilir. Eğer UN*X tipi bir işletim sisteminde çalışıyorsanız, **LC_ALL** çevre değişkenini Türkçe yapıp Python'u yeniden çalıştırmanız yeterli olacaktır. Bunun için bash kabuğu kullanıcıları şu komutu kullanabilirler:

```
$ export LC_ALL=tr_TR
```

Windows üzerinde çalışıyorsanız locale modülüne ihtiyacınız olacak. Modüllerin çağrımı ve kullanımı ile ilgili ayrıntılı açıklamayı ayrı bir bölüm halinde bulacaksınız. Burada sadece küçük bir örnek verilecektir.

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL,'Turkish_Turkey.1254')
'Turkish_Turkey.1254'
>>> a="şeker ve yağ kolestrol yapar."
>>> print a.upper()
ŞEKER VE YAĞ KOLESTROL YAPAR.
```

Yaptıklarımızın örneğini **Şekil-4.1**'de görüyorsunuz.



Şekil-4.1: Python Türkçe'den anlar

UN*X kullanıcılarının program içerisinde yerelleştirme yapmak için birazcık farklı bir yol uygulamaları gerekiyor.

Benzer şekilde, küçük harflere dönüştürme işlemi için **lower()** özelliği kullanılır:

```
>>> b="ÜZÜM ve AĞAÇ"
>>> print b.lower()
üzüm ve ağaç
```

Bir cümlenin sadece ilk harfini büyütmek isterseniz **capitalize()** özelliğinden yararlanabilirsiniz:

```
>>> print a.capitalize()
Şeker ve yağ kolestrol yapar.
```

Program yazarken bazen bir değişkenin büyük harflerden mi, küçük harflerden mi oluştuğunu denetlemeye ihtiyacınız olabilir. O zaman **islower()** ve **isupper()** fonksiyonları imdâdınıza koşacaktır:

```
>>> c="BAĞINIZDA ASMALAR"
>>> d="Boynunuzda Tasmalar"
>>> c.isupper()
1
>>> d.isupper()
0
>>> d.islower()
0
```

Desen Bulma

Cümleler içerisinde desen bulma işlemini aslında **re** ile yapmanızı öneririz, ancak yapacağınız işlem çok basitse birde **re** modülünü hafızaya yüklemek anlamsız olabilir. Desen bulma cümlelerin **find()** özelliği ile gerşekleştirilir ve çoğu zaman bir metnin içerisinde aradığınız bir kelime veya cümlenin var olup olmadığını sınamak için kullanılır. Aşağıda böyle bir örneği görüyorsunuz:

```
>>> cumle='Bu cümlenin içerisinde "sanane" kelimesi vardır.'
>>> cumle.find('sanane')
24
```

Gördüğünüz gibi kelime bulundu ve metin içerisindeki konumu ekrana yazıldı. Burada cümlemiz sayfalar dolusu bir metin olabilirdi. Peki ya sanane kelimesi birden fazla olsaydı ve biz sağdakinin konumunu bulmak isteseydik. Bu durumda **rfind()** fonksiyonunu kullanmamız gerekecekti:

```
>>> cumle='''Bu cümlenin içerisinde "sanane" kelimesi iki kere
vardır. İstersen "sanane" kelimesini sağdan bul.'''
>>> cumle.rfind('sanane')
68
```

Şimdi de aradığımız deseni bulamadığımızda Python'nun davranışına bakalım:

```
>>> cumle.find('banane')
-1
```

Sonucun –1 olması sizi şaşırtmasın. Eğer 0 değeri verilseydi, o zaman aradğımız desen cümlenin ilk karakterinden itibaren başlıyor anlamına gelirdi:

```
>>> cumle='Bu cümlenin içerisinde "sanane" kelimesi vardır.'
>>> cumle.find('Bu')
0
```

Bazen bir cümleyi parçalamak isteyebilirsiniz. Genellikle CGI programcılığında çok kullanılan bu metotla ilgili bir ipucu verelim:

```
>>> HTML='<font color=red><h1>Bu Bir Kırmızı Başlıktır.</h1></font>'
>>> L1=HTML.find('<h1>')+4
>>> L2=HTML.find('</h1>')
>>> print HTML[L1:L2]
Bu Bir Kırmızı Başlıktır.
```

Bu yazılım klasik programlama dillerine aşina olanlar içindir. İyi bir Python programcısı aşağıdaki gibi bir yazılımı benimseyecektir:

```
>>> HTML='<font color=red><h1>Bu Bir Kırmızı Başlıktır.</h1></font>'
>>> print HTML[HTML.find('<h1>')+4: HTML.find('</h1>')]
Bu Bir Kırmızı Başlıktır.
```

Aslında bu tür işlemleri **re** ile yapmanızı tavsiye ederiz.

Desenlere Ayırma ve Birleştirme

Bir cümleyi istediğiniz bir desene göre ayırmanız mümkün. Bu desen genelde boşluk olur ancak, internet uygulamalarında başka desenler de kullanılabilir. Örnek:

```
>>> bunlar="bu kelimesi bu sefer bununla olamaz"
>>> bunlar.split('bu')
['', ' kelimesi ', ' sefer ', 'nunla olamaz']
```

Ayırma işlemi yapıldığında sonuç genellikle bir listeye atanır. Eğer **split()** fonksiyonuna herhangi bir argüman vermezseniz, cümleniz boşluklara (boşluk, satır sonu veya tab) göre ayrılacaktır:

```
>>> "ali    veli hasan".split()
['ali', 'veli', 'hasan']
```

Birden çok cümleyi belirli bir desene göre birleştirebilirsiniz. Yazılımı şu şekildedir:

birlestirmedeseni.join([birlestirilecekliste])

Örneğin:

```
>>> ' ve '.join(['bu','biz','bebek'])
'bu ve biz ve bebek'
```

Desen Değiştirme

Python'da cümlelerle ilgili sık yapılan işlem bir deseni başkası ile değiştirmedir. Genellikle web programcılığında kullanılan **replace()** özelliğine sistem programcıları da rağbet ediyor:

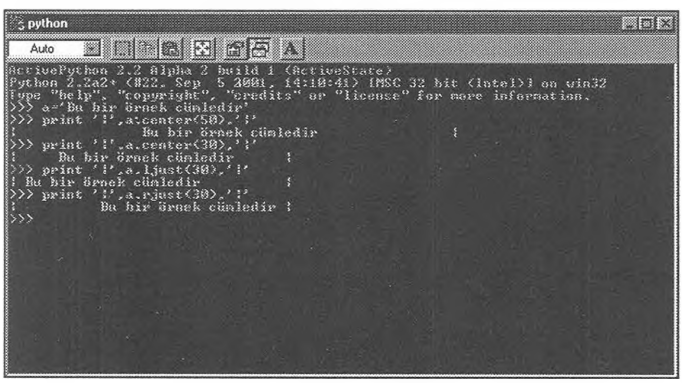
```
>>> a='Çocuklarımı çok seviyorum.'
>>> print a.replace('Çocuklarımı','Bebeklerimi')
Bebeklerimi çok seviyorum.
```

Ortalama

Metin işlemede çoğu zaman, cümleyi belirli bir uzunluk içerisinde ortalamak, sola veya sağa yaslamak gerekir. Yine web programcılarının çok işine yarayacak **just()**, **ljust()** ve **rjust()** özellikleri sizi birçok gereksiz kod yazımından kurtarır. İşte örnekler:

```
>>> a='Bu bir örnek cümledir.'
>>> print '|',a.center(50),'|'
|           Bu bir örnek cümledir.  |
>>> print '|',a.center(30),'|'
|     Bu bir örnek cümledir.        |
>>> print '|',a.ljust(30),'|'
| Bu bir örnek cümledir.            |
>>> print '|',a.rjust(30),'|'
|           Bu bir örnek cümledir.  |
```

Bu örnekleri **Şekil-4.2**'de görüyorsunuz



Şekil-4.2: Eğer bir kelime işlemci yazacaksanız, metin ortalama komutları hazır!

Eğer bu işlemleri web programı içerisinde yapıyorsanız, boşlukların tam anlamı ile grünebilmesi için onları ** ** ile değiştimrniz gerkemektedir:

```
>>> ortalanmis=a.center(30)
>>> print ortalanmis.replace(' ',' ')
    Bu bir örnek 
cümledir     
```

Ya da daha nesnel bir yaklaşımla:

```
>>> print a.center(30).replace(' ',' ')
    Bu bir örnek 
cümledir     
```



Python 1.6'dan önce cümle özellikleri yoktu ve bu string modülü ile işlemler yapılırdı. Eğer Python1.5.2 kullanıyorsanız (Muhtemelen Linux işletim sistemlerinde python yazıldığında bu sürüm çalışır.) cümle nesnelerini kullanamazsınız. Linux kullanıyorsanız Python2 şeklinde çağırmalısınız. Eski programlara uyumluluk açısından Python'un yeni sürümlerinde de string modülü vardır. Bir örnek vermek gerekirse:

```
>>> import string
>>> a="Bu cümlede nereyi değiştirelim."
>>> print string.replace(a, "nereyi", "burayı")
Bu cümlede burayı değiştirelim.
```

Kaçış Karakterleri

Python ve diğer programlama dillerinde bazı karakterlerin özel anlamları vardır. Bu karakterlere "kaçış karakteri" denir. Bunlardan sık kullanılanları şunlardır:

Kaçış karakteri	Anlamı
\	Satırın devam ettiğini gösterir. Bazen Python'da bir satır öyle uzun olur ki, okunabilirliği artırmak için ikinci satıra geçmek gerekebilir. Bu durumda bu karakteri kullanarak, aslında ikinci (ya da üçüncü, dördüncü...) satırın birinci satırın devamı olduğunu belirtebilirsiniz.
\\	Gerçek anlamda \ karakteri olduğunu belirtir. Çünkü yalnız başına \, sonraki gelen karakterin kaçış karakteri olarak kullanılacağını belirtir.
\'	' karakteri. Örneğin: neden='Mustafa\'nın isteği'
\"	" karakteri. Örneğin: kim="Arkadaşım \ "Ankaralı'yım\" dedi"
\a	Bilgisayarınızın "beep" sesi çıkarmasını sağlar. Örneğin print "\a"
\b	geriye doğru silme
\e	"escape" karakteri
\n	satır sonu
\v	dikey tab
\t	yatay tab

Denetleme Deyimleri

05

Python'da program akışını denetleyen deyimleri bu bölümde inceleyeceğiz. Aslında bu bölümle birlikte, gerçek programlamaya ilk adımı da atmış olacağız. Bu bölümün ilk kısımlarının biraz sıkıcı gelebileceğini şimdiden belirtelim.

Önce Python karşılaştırma operatörlerinden bahsedeceğiz, daha sonra karşılaştırma deyimi ve döngü deyimlerini öğreneceğiz.

Karşılaştırma Operatörleri

Python'da belli başlı 6 tane karşılaştırma operatörü vardır. Karşılaştırma operatörleri sayılar, cümleler, listeler, tüpler ve sözlükler için geçerlidir. Karşılaştırma operatörleri şunlardır:

Karşılaştırma Operatörü	Yaptığı Karşılaştırma
>	büyüklük
>=	büyüklük ya da eşitlik
<	küçüklük
<=	küçüklük ya da eşitlik
==	eşitlik
<>	eşit değil

Karşılaştırma doğru ise sonuç 1, yanlış ise 0 olarak dönecek, böylelikle programımızın akışına karar verilecektir.

Şimdi hemen örneklere bakalım.

```
>>> a > b
0
>>> b > a
1
>>> 5 > 7
0
>>> 3 < 5
1
>>> 'elma' > 'armut' #Cümle uzunluğu karşılaştırılıyor
1
>>> 6 >= 5
1
>>> 5 >= 5
1
>>> 5 <= 5
1
>>> 5 <= 9
1
>>> 5 == 9
0
>>> 7 == 7
1
>>> 'kalem' == 'kalem'
1
>>> 5 != 5
0
>>> 5 != 1

>>> a=('t',5)
>>> b=('t',6)
>>> a == b  #iki tüpün eşit olup olmadığı karşılaştırılıyor.
0
>>> b=('t',5)
>>> a == b
1
```

Mantıksal Operatörler

Birden fazla karşılaştırma yapılacaksa mantıksal opertörler kullanılabilir. Mantıksal operatörler şunlardır:

Mantıksal Operatör	Anlamı
and	ve
or	veya
not	değil

Mantıksal operatörler için doğruluk tablosu şöyledir:

A	Operatör	B	Sonuç
doğru	and	doğru	doğru
doğru	and	yanlış	yanlış
yanlış	and	doğru	yanlış
yanlış	and	yanlış	yanlış
doğru	or	doğru	doğru
doğru	or	yanlış	doğru
yanlış	or	doğru	doğru
yanlış	or	yanlış	yanlış
	not	yanlış	doğru
	not	doğru	yanlış

Kısaca özetlersek, sonucun doğru olabilmesi için; **and** kullandığımızda her ikisinin de doğru olması gerekir, **or** kullanırsak sadece birisinin doğru olması yeterlidir. Değil (**not**) doğruyu yanlışa, yanlışı da doğruya çevirecektir. Biz burada doğru ve yanlış kavramlarını kullandık, bilgisayar dilinde doğru yerine 1, yanlış yerine 0 kullanılır. Python'da sıfırdan büyük bir tamsayı doğru kabul edilir. Aynı şekilde boyu sıfırdan büyük cümleler, elemanı olan listeler ve tüpler herhangi bir anahtarı olan sözlükler de doğru sayılır.

if deyimi

Eğer anlamına gelen **if**, sonuç doğru olduğunda sonra gelen bloğun çalıştırılmasını sağlar. Bloktan kastımız geçerli Python nesnelerinden oluşan gruptur.

Yazılışı şöyledir:

if karşılaştırma:
karşılaştıma doğruysa
yürütülecek blok

Örneğin:

```
>>> if 5 > 4:
    print "Beş dörtten büyük"

Beş dörtten büyük
```

Burada ilk kez bir blok görüyorsunuz. İyi de bloğun başlangıç ve bitişi nasıl belirtiliyor? Perl, Php, Java, C gibi dillerde {} ile, Pascal'da begin, end ile belirtilirken, Python'da bu tür gereksiz belirteçlere ihtiyaç duyulmaz. Bloğun başlangıç ve bitişi girinti ve çıkıntılarla belirlenir. Bir bloğu üst üste iki nokta (:) ile başlatırız. Bloğun tüm satırları, blok başlamadan bir önceki satırdan daha içerde yazılır. Eğer bloğumuz tek bir satırdan oluşuyorsa o zaman iki nokta üst üsteden sonra devam edebiliriz. Yukarıdaki örneği şu şekilde de yazabilirdik:

```
>>> if 5 > 4: print "Beş dörtten büyük"

Beş dörtten büyük
```

Programınız çok uzunsa, bir veya iki satırdan oluşabiliecek blokları bu şekilde yazmak hem programınızın okunabilirliğini artırır hem de kodunuz daha kısa olur, örneğin:

```
>>> if 1: print "Doğru";a=9

Doğru
```

Dikkatinizi çektiyse karşılaştırmaları yaparken parantez de () kullanmadık. Bu da Python yazılımını kısaltan diğer bir özellik. Elbetteki kullansanız da buna itiraz etmez:

```
>>> if (5 > 4): print "Beş dörtten büyük"
...
Beş dörtten büyük
```

Ancak okunuşu basitleştirmek için parantezleri kullanmamanızı tavsiye ederiz. Eğer birden çok karşılaştırmayı mantıksal operatörler ile birleştireceksenız okumayı kolaylaştırmak amacıyla parantezleri kullanmanızda yarar vardır fakat bunun dışında gereksizdir:

```
>>> if (5 > 4) or ('elma'=='erik'):
print 'Sonuç Doğru'
...
Sonuç Doğru
```

Programı ile

```
>>> if 5 > 4 or 'elma'=='erik':
...     print 'Sonuç Doğru'
...
Sonuç Doğru
```

programı birbirine eştir. Değil operatörünü de şu şekilde kullanabilirsiniz:

```
>>> if not  5 < 4 : print "Beş dörtten büyük değil"
...
Beş dörtten büyük değil
```

in Deyimi

Klasik bir programcı mantığı ile bir liste içerisinde herhangi bir elemanın var olup olmadığını sınamak için şöyle bir kod yazmak gerekir:

```
>>> L=['Fatih','Melike','Dilek']
>>> var=0
>>> for i in L:
                if i=='Melike': var=1
```

```
>>> if var:
                    print '"Melike" L listesinde bulunuyor.'
```

Fakat Python mantığı ile düşünen bir programcı bunun daha kolay yapılabilmesi gerektiğini bilir ve şöyle bir kod yazar:

```
>>> if "Melike" in L: print '"Melike" L listesinde bulunuyor.'

"Melike" L listesinde bulunuyor.
```

Gördüğünüz gibi **in** deyimi aradığınız terimin listenin içerisinde olup olmadığına bakar.

Bu deyim cümleler için de kullanılabilir. Varlığı araştırılan nesne sadece ve sadece bir karakter olabilir:

```
>>> if "l" in "kelebek": print "'l' var"
...
'l' var
```

if-else

if ile birlikte kullanılan **else, if**'deki karşılaştırma doğru olmadığında ardından gelen bloğun yürütülmesini sağlar. Yazılışı şöyledir:

if karşılaştırma:
    karşılaştırma doğruysa
    yürütülecek blok
else:
    karşılaştırma yanlışsa
    yürütülecek blok

Şimdi gerçek anlamda Python programı yazarak **if-else**'in nasıl kullanıldığını görelim. PythonWin programını çalıştırıp File/New yolunu izledikten sonra gelen pencerede Python script seçeneği aktifken tamam düğmesine tıklayarak yeni bir pencere açın ve aşağıdaki satırları yazın (Koyu ve yatık yazılmış ilk satırı yazmayın. Kitapta geçen program kodlarından önce görece-

ğiniz bu satır, program ismini göstermektedir. Bu programları *http://mbaser.kolayweb.com/pythonkitabı* adresinde bulabilirsiniz).

UN*X kullanıcıları ise idle geliştirme ortamını veya sık kullandıkları bir metin düzenleyicisi açıp program kodlarını yazabilirler. Tercih ettikleri bir düzenleyici yoksa Nedit'i önerebiliriz.

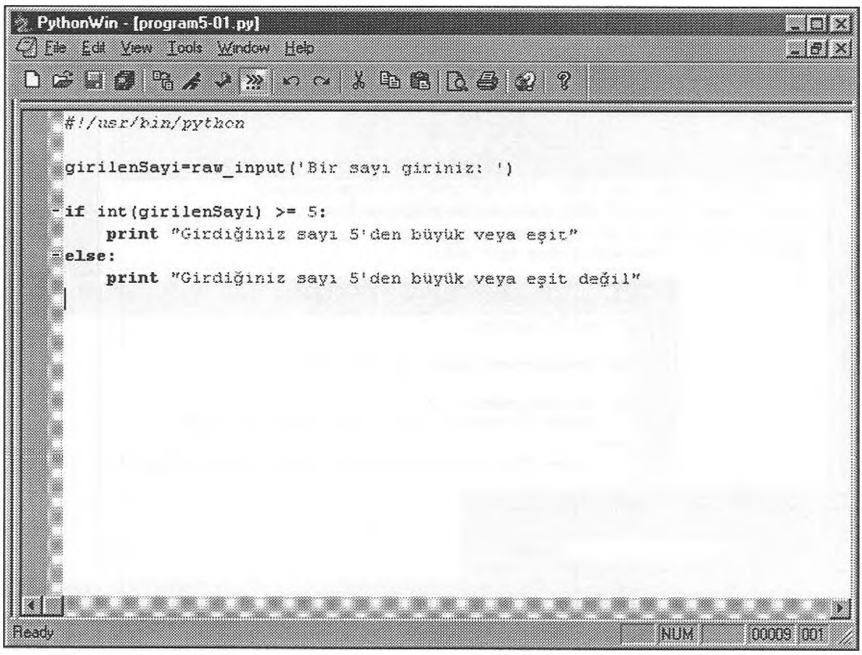
program5-01.py

```
#!/usr/bin/python

girilenSayi=raw_input('Bir sayı giriniz: ')

if int(girilenSayi) >= 5:
    print "Girdiğiniz sayı 5'den büyük veya eşit"
else:
    print "Girdiğiniz sayı 5'den büyük veya eşit değil"
```

Yazdığınız programı File/Save yolunu izleyerek **program5-01.py** ismiyle kaydediniz. Kaydedilmiş programın örneğini **Şekil-5.1**'de görüyorsunuz.



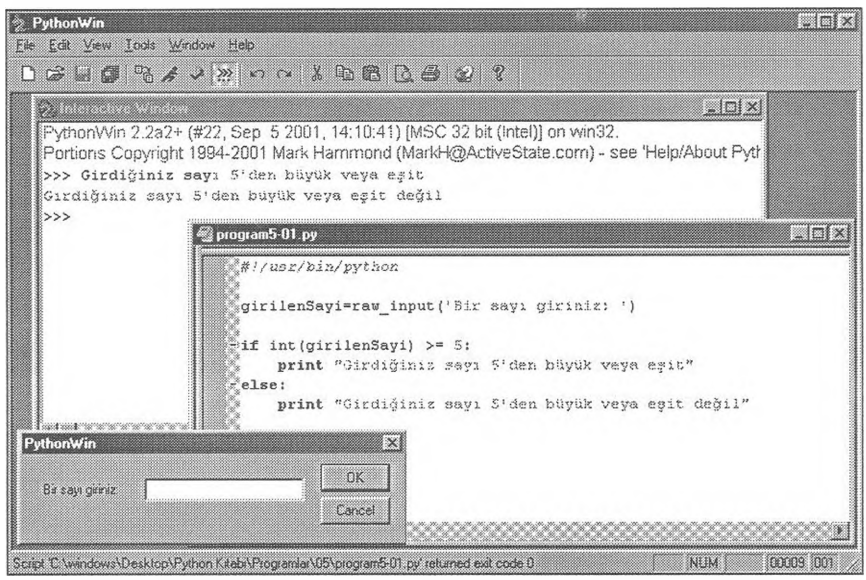
Şekil-5.1: İlk Python programınız

Program satırlarını anlatmadan önce, nasıl çalıştırabileceğimizi öğrenelim. Önce Windows kullanıcılarının programı PythonWin aracında nasıl çalıştırabileceklerine bakalım, sonra UN*X tipi işletim sistemlerinde nasıl çalıştıracağımızı göreceğiz.

PythonWin'de **File/Run** script yolunu izleyerek yazdığınız bir programı çalıştırabilirsiniz. Kısayol için F5 tuşunu kullanmanız yeterli. Bunlardan birisini yaptığınızda programınız çalıştırılacaktır. Eğer standart girdiden birşeyler okuyorsanız PythonWin bunu yeni bir pencerede isteyecektir. Sonuçlar ise etkileşimli kabuk içerisinde görüntülenecektir. Bilgi giriş ve çıkışları bu kabuk ile denetlenir. Bizim programımızda bir bilgi girişi istendiğinden hemen etkileşimli kabuktan bir değer giriniz. Aşağıda verdiğimiz program girdi ve çıktısının örneğini **Şekil-5.2**'de görüyorsunuz:

```
Bir sayı giriniz: 86
Girdiğiniz sayı 5'den büyük veya eşit

>>>
Bir sayı giriniz: 3
Girdiğiniz sayı 5'den büyük veya eşit değil
```



Şekil-5.2: Etkileşimli kabukta programın girdi ve çıktısı denetlenir.

Programınızı **program5-01.py** olarak kaydettiğinizi varsayarak UN*X kabuğundan nasıl çalıştırabileceğinizi görelim:

Önce programa çalıştırma yetkisi vereceğiz:

```
chmod +x program5-01.py
```

Şimdi de bu programı çalıştıralım:

```
./ornek1.py
```

Şimdi programı satır satır açıklamaya çalışalım.

Uzantıyı neden **.py** olarak seçtik? Bu sorunun cevabı oldukça basit. Tüm Python programlarının uzantısı **.py** olmalıdır. Aslında UN*X tipi bir işletim sistemi kullanıyorsanız ve eğer yazdığınız program başka bir program tarafından içerilmeyecekse (bir modül değilse) uzantının pek bir anlamı yok. Ancak Python programları çoğunlukla modüler halde yazılır ve yazdığınız kod başka bir program tarafından içerilir. Python ön tanımlı olarak sadece **.py** uzantılı dosyaları çağırır. Windows kullanıyorsanız, işletim sistemi dosyanın hangi program tarafından yürütüleceğine dosyanın uzantısına bakarak karar verir. Windows tüm **.py** uzantılı dosyaları Python yorumlayıcısına göndererek dosyaların çalıştırılmasını sağlar. Windows'da programınızın üzerine çift tıkladığınızda program çalıştırılır.

Windows kullanıcıları CGI programlarını yürütürken web sunucusu olarak Apache kullanmayacaksa, programın ilk satırı onlar için anlamsızdır. Apache kullanacak Windows kullanıcıları ilk satırı Python'nun kurulu olduğu patika ile değiştirmelidirler. Şu anda çalıştığınız makine üzerinde bu satırı şununla değiştirmekte fayda var:

```
#!c:\\python22\\python.exe
```

Python kodlarında **#** ile başlayan tüm satırlar yorum satırlarıdır. Yani Python bu satırları göz ardı eder. Bunlar programı yazan (ya da ilerde okuyacak olanlar) için açıklamaların bulunduğu satırlardır. UNIX tipi işletim sistem-

lerinde, derlenmemiş bir dosya çalıştırılabilme hakkına sahip ise, sistem ilk satıra bakarak geri kalan kodun kimin tarafından yorumlanacağına karar verir. Bizim ilk satırımızdaki ilk iki karakteri atarsak, bu kodun **/usr/bin** patikasındaki **python** tarafından yürütüleceği anlaşılır. Eğer Python sisteminizde başka bir patikada bulunuyorsa program çalıştırılamayacaktır. Python'un nerede bulunduğunu öğrenmek için daha önce de belirttiğimiz gibi:

```
$ which python
```

komutunu çalıştırın. Eğer ilk satırı **"#!/usr/bin/env python"** değiştirecek olursanız, Python yorumlayıcısının nerede olduğunu bilmenize gerek kalmaz ve tüm Python kurulu UN*X platformlarında çalışır.

```
girilenSayi=raw_input('Bir sayı giriniz: ')
```

satırında ise iki işlem yapılıyor. **Raw_input()** hazır fonksiyonu klavyeden veri girişi yapmak için kullanılır (daha doğru bir deyişle standart girdiden). Yazılım şöyledir:

*girdininAtanacagiDegisken=**raw_input**(girdi_baslamadan_once_ekrana_yazılacak_cumle)*

Bu satır ile klavyeden bir girdi isteniyor ve girilen değer **girilenSayi** değişkenine atanıyor. Klavyeden kullanıcının girdiği değerler cümle olarak algılanılırlar.

Daha sonra bir **if-else** başlatılıyor. Karşılaştırma yapılmadan önce girilen değer tamsayıya çevrilmedir. Aksi taktirde cümlenin ilk karakterinin ascii kodundaki değer ile karşılaştırma yapılacaktır. Eğer kullanıcı 5'den büyük bir sayı ya da 5 sayısını girmişse **girilenSayi >= 5** karşılaştırması doğrulanıyor ve

```
print "Girdiğiniz sayı 5'den büyük veya eşit"
```

bloğu işletiliyor. Kullanıcı daha küçük bir sayı girerse karşılaştıma yanlış oluyor ve **else** bloğu işletilerek ekrana "Girdiğiniz sayı 5'den büyük veya eşit değil" yazdırılıyor.

Böylelikle ilk Python programınızı yazmış ve işletmiş oldunuz.

if-elif-else

if deyiminin bir diğer çeşidi de bir değer için birden çok karşılaştırmanın yapılacağı **if-elif-else**'dir. Pascal, C ve diğer bazı dillerde **switch** veya **case** deyimi varken Python'da bu deyime rastlamıyoruz. Bunun yerine **if-elif-else** kullanıyoruz. Bu deyimle menü oluşturmanızı önermiyoruz, çünkü menü oluşturmanın daha etkin, kolay ve hızlı yolları vardır. Bunu da ilerde göreceğiz. Yazılışı aşağıdaki gibidir:

if *karşılaştırma1:*
    karşılaştırma1 doğruysa
    yürütülecek blok
elif *karşılaştırma2:*
    karşılaştırma2 doğruysa
    yürütülecek blok
elif *karşılaştırma3:*
    karşılaştırma3 doğruysa
    yürütülecek blok
.
.
.
else:
    karşılaştırmaların hepsi
    yanlışsa yürütülecek blok

Hemen yukarıda verdiğimiz örneği yeniden düzenleyelim:

program5-02.py

```
#!/usr/bin/python

girilenSayi=int(raw_input('Bir sayı giriniz: '))

if girilenSayi > 5:
     print "Girdiğiniz sayı 5'ten büyük"
elif girilenSayi == 5:
     print "Girdiğiniz sayı 5"
```

```
else:
     print "Girdiğiniz sayı 5'ten küçük"
```

Bu program klavyeden girilen sayının 5'ten büyük, 5'e eşit ya da 5'ten küçük (diğer iki koşul doğru değilse geriye kalan koşul) olup olmadığını denetliyor. Bu gibi programlarda blokları iki nokta üstüstenin yanına yazarak okunurluğu artıabilirsiniz:

```
if girilenSayi > 5: print "Girdiğiniz sayı 5'ten büyük"
elif girilenSayi == 5: print "Girdiğiniz sayı 5"
else: print "Girdiğiniz sayı 5'ten küçük"
```

for Deyimi

Python'daki iki döngü deyiminden biri olan **for**'dan bahsedeceğiz. Bir programın vazgeçilmez deyimlerinden birisi de döngüdür. Hemen hemen her programda birkaç tane döngü bulunur. Peki döngü nedir? Aynen başınızın dönmesi gibi birşey. Bir listenin elemanlarının teker teker ele alınmasını döngü olarak niteleyebiliriz. Python'da for deyimi sadece cümleler, listeler ve tüpler üzerinde yapılır. Bunu Perl'deki **foreach** deyimine benzetebilirsiniz. Yazılım aşağıdaki gibidir:

for *degisken* ***in*** *liste:*
  for deyimi için
  işletilecek blok

Hemen bir örnek verelim:

```
>>> for x in [1,2,3,4,5]:
print 'x değişkeninin şimdiki değeri: ', x
...
x değişkeninin şimdiki değeri:  1
x değişkeninin şimdiki değeri:  2
x değişkeninin şimdiki değeri:  3
x değişkeninin şimdiki değeri:  4
x değişkeninin şimdiki değeri:  5
```

Yukarıdaki örnekte listemizin elemanları 1-5 arası tamsayılardı. Liste elemanları herhangi bir Python nesnesi olabilir. Aşağıdaki örnekde bunu görüyorsunuz:

```
>>> for x in ['Dilek', {'meyveler':['elma','armut']}]:
...     print 'x değişkeninin şimdiki değeri: ', x
...
x değişkeninin şimdiki değeri:  Dilek
x değişkeninin şimdiki değeri:  {'meyveler': ['elma', 'armut']}
```

Peki 0-100 arası sayılar için bir döngüye ihtiyacımız varsa? Bu listeyi teker teker mi yazacağız? Tabiiki hayır: **range()** hazır fonksiyonu tamsayılardan bir liste oluşturmak için kullanılanılır. Yazılımı şöyle:

range(baslangic, bitis, artis_miktari)

Eğer ilk argüman verilmezse başlangıç sıfır alınır, son argüman verilmezse artış miktarı bir 1 olur. Örneğin:

```
>>> range(1,40,3)
[1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37]
```

for döngüsü ile birlikte şöyle kullanabiliriz:

```
>>> for i in range(1,5): print i
...
1
2
3
4
```

Yukarıda yazdığımız döngü, C/C++ programlama dillerinde şu şekilde yazılabilirdi (i'nin daha önceden tamsayı olarak tanımlandığını varsayalım):

```
for (i=1;i<6;i++)  {
                   print (i);
}
```

i değişkeninin önüne bir $ işareti koyarsanız, aynı döngünün Perl eşleniğini elde etmiş olursunuz. Bir kez daha Python'un ne kadar kolay olduğunu ve kod yazılımının çok kısa olduğunu görmüş oldunuz.

Burada dikkatinizi çekmek istediğimiz nokta bitiş sayısının liste elemanı olmayışıdır. Yani son eleman listelenmez.

len() ve **range()** fonksiyonlarını şu şekilde birlikte kullanabilirsiniz:

```
liste=["gorkem", "mustafa", "akgun", "şakir"]
>>> liste=["gorkem", "mustafa", "akgun", "şakir"]
>>> for i in range(len(liste)):
...     print i, liste[i]
...
0 gorkem
1 mustafa
2 akgun
3 şakir
```

Bir sözlük için döngüyü şöyle yazabiliriz:

```
>>> sozluk={'adı':'Mustafa', 'soyadı':'Başer', 'no':542407}
>>> for a in sozluk.keys():
...     print a, ':', sozluk[a]
...
adı : Mustafa
no : 542407
soyadı : Başer
```

Bu program parçasında, döngü aslında sozluk değişkeninin anahtarlarından oluşan bir liste üzerinde gerçekleşti. Önceki bölümlerden anımsayacak olursak **sozluk.keys()**, sozluk'un anahtarlarını bir liste halinde verir.

Eğer Python2.1'den daha yeni bir sürüm kullanıyorsanız, for döngüsü sözlükler üzerinde iterasyon olarak çalışabilir. Yani yukarıdaki program parçasını daha da kısaltarak:

```
>>> for a in sozluk:
...             print a,':', sozluk[a]
...
adı : Mustafa
no : 542407
soyadı : Başer
```

Şeklinde de yazabilirdik. Hangisini kullanacağınıza siz karar verebilirsiniz. Tavsiyemiz ikinci yolu kullanmanızdır.

Cümleler üzerinde for döngüsünü oluşturmanız olası:

```
>>> for i in "Aşkın gözü kördür": print i , "-",
...
A - ş - k - ı - n -  - g - ö - z - ü -  - k - ö - r - d - ü - r -
```

Bir dosyanın içeriğini satır satır ekrana bastırmak için for deyimini şu şekilde kullanabilirsiniz:

```
>>> dosya=open("/etc/passwd")
>>> for satir in dosya.readlines():
...     print satir
>>> dosya.close()
```

Ya da Python2.1 ve daha üzeri sürümlerde, iterasyon yaparak:

```
>>> for satir in open("/etc/passwd"):
...     print satir
```

Python'da kolaylıklar bitmiyor.

break

for veya **while** döngüsünden herhangi bir zamanda çıkmak istediğinizde **break** hazır fonksiyonunu kullanmalısınız. **break** fonksiyonu döngüyü olduğu yerde keser; program, döngü sonrası blokların yürütülmesi ile devam eder. Aşağıda kullanıcıdan beş tane sayı girilmesi isteniyor. Eğer girilen değer "ç" ise döngü hemen sonlandırılıyor:

program5-03.py

```
#!/usr/bin/python

for x in range(5):
    girilenSayi=raw_input('Bir harf ya da rakam giriniz: ')
    if girilenSayi == 'ç': break
    print "Girdiğiniz değer: ", girilenSayi
```

Şimdi yukarıdaki programı çalıştıralım:

```
Bir harf ya da rakam giriniz: o
Girdiğiniz değer:  o
Bir harf ya da rakam giriniz: ç
```

Eğer ikinci seferde ç harfini girmeseydik tekrar bir harf ya da sayı girmemizi isteyecekti. Bu döngü ya ç harfi girene kadar devam edecek ya da 5 kere tekrarlanacaktı.

for-else

Bazılarınızın "bu da nesi" dediğini duyar gibi oluyorum. **else**'in bir döngü ile kullanılması gerçekten ilginç. Peki ne işe yarar? Cevap: hayatımızı kolaylaştırmaya... Bir döngüde else bloğu, eğer o döngü normal olarak bitmişse işletilir. Break ile döngü sonlandırılırsa bu kısım yürütülmez. Yukarıdaki örneğe bir else ekleyelim:

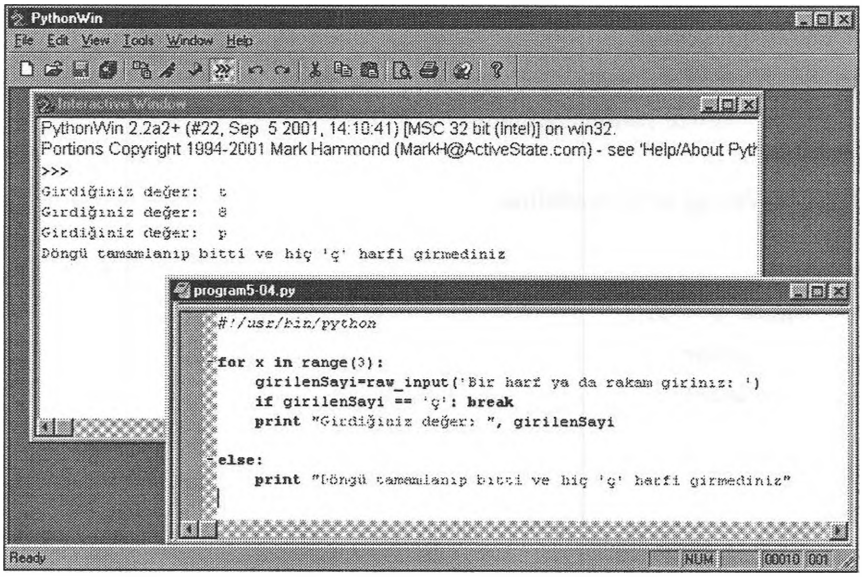
program5-04.py

```
#!/usr/bin/python

for x in range(3):
    girilenSayi=raw_input('Bir harf ya da rakam giriniz: ')
    if girilenSayi == 'ç': break
    print "Girdiğiniz değer: ", girilenSayi

else:
    print "Döngü tamamlanıp bitti ve hiç 'ç' harfi girmediniz"
```

Bu programı iki kez çalıştıralım. Birincisinde hiç "ç" harfi girmeyelim. İkinci seferde ise hemen "ç" ile döngüyü sonlandıralım. Sonucu **Şekil-5.3**'de görüyorsunuz.



Şekil-5.3: For ve else birlikte kullanılabilir.

continue

continue deyiminin bir döngü içerisindeki etkisi, döngünün bir sonraki elemanının işletilmesi ile devam etmesidir:

```
>>> for i in range(5):
...     if i==2: continue
...     else: print i
...
0
1
3
4
```

Bu örnekte gördüğünüz gibi **i** değişkeni **2**'ye eşit olduğunda döngü bir sonraki adıma geçti.

while Deyimi

Python'da kullanılan diğer bir döngüdür. Bir koşul doğruluğunu devam ettirdiği sürece döngü devam eder. Yazılımı şöyledir:

while *karşılaştırma:*
karşılaştırma doğru olduğu
sürece yürütülecek blok

Basit bir örneği şöyle verebiliriz.

```
>>> i=0
>>> while i < 5:
...     print i
...     i=i+1
...
0
1
2
3
4
```

Yukarıdaki **program5-03.py** isimli programı while ile yazarsak:

program5-05.py

```
#!/usr/bin/python

girilenSayi=0
i=0
while girilenSayi <> 'ç' and i < 5:
        girilenSayi=raw_input('Bir harf ya da rakam giriniz: ')
        print "Girdiğiniz değer: ", girilenSayi
        i=i+1
```

Else ve break deyimleri while ile birlikte de kullanılabilir.

Bir dosyanın tamamını **while** deyimiyle de okuyabilirsiniz. Aşağıdaki program "/etc/services" dosyasının içeriğini ekrana yazdırıyor.

```
>>> satir=1
>>> dosya=open("/etc/services")
>>> while satir:
...     satir=dosya.readline()
...     print satir
>>> dosya.close()
```

Burada **satir=1** tanımlamasının neden yapıldığını düşününüz.

İleri for

Python **for** deyiminin birden fazla döngü değişkenini aynı anda çevirmenize fırsat tanır. Kod yazılımını oldukça kısaltan ve anlaşılırlığı artıran ileri **for** deyimini kullanmanız, sizin iyi bir programcı olduğunuzun da göstergesidir. Şimdi kısa bir örnek vererek bu bölümü sonlandıralım:

```
>>> email=[('Mustafa', 'mbaser@kolaymail.com'),
('Görkem', 'gorkem@gelecek.com.tr')]
>>> for isim, adres in email:
...     print isim,':',adres
...
Mustafa : mbaser@kolaymail.com
Görkem : gorkem@gelecek.com.tr
```

Eğer aynı işi, klasik programcı mantığıyla yapsaydınız bakın kodunuz ne kadar uzayacak anlaşılması ne kadar zor olacaktı:

```
>>> email=[('Mustafa', 'mbaser@kolaymail.com'),
'Görkem', gorkem@gelecek.com.tr')]
>>> for i in range(0, len(email)):
...     print email[i][0], ':', email[i][1]
...
Mustafa : mbaser@kolaymail.com
Görkem : gorkem@gelecek.com.tr
```

Fonksiyonlar



Fonksiyonlar programların vazgeçilemez bölümleridir. Python'da fonksiyon tanımlamak da çağırmak da diğer dillere göre oldukça kolaydır. Özellikle istediğiniz veri tipini bir fonksiyona argüman olarak gönderebilmeniz ve geri dönüşü de yine istediğiniz veri tipinde olabilmesi, Python'un olağanüstü bir özelliğidir. Bu bölümde birer Python nesnesi olan fonksiyonlara göz atacağız.

Fonksiyon Nedir?

Programları yazarken bazı durumlarda belirli bir işi sürekli yapmak gerekebilir. Her seferinde o kodları tekrar yazmak yerine belirli bir grup altında toplayıp, kodların istediğimiz zaman çalıştırılmasını sağlayabiliriz. İşte bunlara fonksiyon denir. Fonksiyonlar genelde belirli bir işi yapmak üzere yazılmış kodlardır ve bir programın vazgeçilmez unsurlarıdır. İyi yazılmış bir program fonksiyonların birleşmesinden meydana gelir. Bu fonksiyonlar ise ilerde göreceğimiz gibi sınıfların içerisine, sınıflar da modüllerin içerisine gömülürler.

Fonksiyonlara çoğu zaman işletilmeden önce bir bilgi girişi yapılır, işlenen bilgi çağırıcısına geri gönderilir. Ancak bazen bilgi girişi almayan ya da geriye bilgi vermeyen fonkiyonlar da olabilir. Çoğu programlama dilinde procedure ve fonksiyon gibi iki tür varken, Python'da hepsi fonksiyon olarak adlandırılır.

Daha önceki bölümlerde hazır fonksiyonları kullandık. Örneğin bir cümleyi tamsayıya çeviren **int()** fonksiyonu. Bu fonksiyon kendisine verilen cümle değişkeni, tamsayıya çevirir, giren bilgi bir cümle, geri dönen bilgi ise bu cümleye karşılık gelen tamsayıdır. Kullanımı şöyleydi:

```
cumle='1970'
tamSayi=int(cumle)
```

Fonksiyonlara verilen bilgilere genel olarak argüman denir. Yukarıdaki örneğimizde cümle değişkeni **int()** fonksiyonuna verilen bir argümandır. Bazı fonksiyonlara birden çok argüman verilebilir. Bazıları ise hiç argüman istemezler. Nesnelere ait fonksiyonlar da vardır, bunların birçoğunu geçtiğimiz bölümlerde öğrendik. Örneğin cümle nesnesinde bulunan **split()** fonksiyonu bir tane argüman istemektedir:

```
>>> 'mbaser@kolaymail.com'.split('@')
```

Burada **split()** bir nesne fonksiyonudur, yaptığı iş ise, adresi kullanıcı adı ve alan olmak üzere ikiye ayırmaktır. Burada geriye dönen bir listedir ve bu listede iki tane eleman vardır. Eğer birden çok bilgi geri dönecekse bu bilgiler genelde tüp, liste veya sözlük içerisinde geriye dönerler.

Bir fonksiyona birden fazla argüman veriliyorsa bunlar virgül ile ayrılır:

```
>>> 'Dün sınıfta vardı.'.replace('vardı','yoktu')
```

Bazı fonksiyonlar çağırana bir bilgi dönüşü yapmaz. Buna verebileceğimiz örnek ise bir dosyaya bilgi aktarmak için kullanılan **write()** fonksiyonudur. Bu fonksiyon kendisine bir argüman ile sağlanan bilgiyi dosya nesnesine yazar, fakat geriye bir şey dönmez. Bir de hiç argüman almadığı halde geriye bilgi dönüşü yapan fonksiyonlar vardır. Bu tür fonksiyonlara şu örneği verebiliriz:

```
>>> time.time()
```

Son olarak ne bilgi isteyen, ne de bilgi dönüşü yapan fonksiyonlara bir örnek verelim: açılan bir dosyayı kapatan **close()** fonksiyonu... Özet olarak fonksiyonlar dört tiptir:

- bilgi girişi ve dönüşü olmayan
- bilgi girişi olup geriye bir şey döndürmeyen
- bilgi girişi olmayıp geriye bilgi döndüren
- bilgi girişi ve dönüşü olan

Yukarıda verdiğimiz örneklere dikkat ederseniz her biri sadece bir işi yapmak üzere hazırlanmıştır. İyi bir programcı fonksiyonları tasarlarken sadece bir işi yaptırmayı hedefler.

Bu verdiğimiz örnekler Python içerisinde hazır bulunan bazı fonksiyonlardır. Ancak her istediğiniz işi yapan hazır bir fonksiyon bulunmaz. Bu durumda kendi fonksiyonunuzu yazmanız gerekir. Bu tür fonksiyonlar "kullanıcı fonksiyonu" olarak adlandırılır. Bu bölümde bunları göreceksiniz.

Python birçok hazır fonksiyon ile birlikte gelmektedir. Mümkün olduğu kadar bu hazır fonksiyonlar kullanılmalıdır. Bu hem işinizi kolaylaştırır hem de programınızın hızını artırır. İşinize yarayacak bir fonksiyon bulamadığınız zaman mecburen kendi fonksiyonunuzu tanımlamak zorunda kalırsınız. Zaten kendi tanımladığınız fonksiyonlar içerisinde de Python hazır fonksiyonlarını kullanacaksınız.

Fonksiyonların Tanımlanması ve Çağrılması

Python'da fonksiyonlar **def** ifadesi ile tanımlanırlar. Genel yazılışı şöyledir:

def *fonksiyon_adi(arguman_listesi):*
    Fonksiyon içerisinde
    işletilecek blok

Kullanıcı fonksiyonlarının çağrılması da hazır fonksiyonlardaki gibidir. Şu şekilde çağrılırlar:

```
degisken=fonksiyon_adi(arguman_listesi)
```

Eğer bir bilgi dönüşü yoksa değişken yazılmayabilir, yazılsa da değeri "None" olacaktır. Fonksiyonumuz bilgi girişi istemiyorsa arguman_listesi yazmaya gerek yoktur.

Programlama diline yeni başlayan okuyucularımızın fonksiyonları daha iyi anlayabilmesi için yukarıda yaptığımız ayırım sırasına göre inceleyeceğiz. Bu bölümde kodlarımızı çoğunlukla bir dosyaya yazarak programlarımızı oluşturacağız.

Bilgi Girişi ve Dönüşü Olmayan Fonksiyonlar

İsterseniz hemen basit bir fonksiyon tanımı ile başlayalım. Tanımlayacağımız ilk fonksiyona ne bilgi girişi olsun ne de fonksiyondan bilgi dönüşü olsun. Yapacağı iş ekrana 'Burası fonksiyon' yazısını yazmak olsun. Böyle bir fonksiyonun tanımlanması ve çağrılmasına şöyle bir örnek verelim:

program6_1.py

```
#!c:\\python22\\python.exe

def ilkFonksiyonum():
      print 'Burası fonksiyon.'

ilkFonksiyonum()
```

Bu programı çalıştırırsak çıktısı aynen şöyle oluyor:

```
Burası fonksiyon
```

Bilgi Girişi Olup Geriye Birşey Döndürmeyen Fonksiyonlar

Kullanacağımız başka bir çeşit fonksiyon da bizden bilgi isteyip bize bir şey vermeyen, adeta sömürücü cinsten olanlardır. Bunlar çağrılırken bizden bilgi isterler, ancak bilgiyi yoğurduktan sonra bize hiçbir şey vermeyip kendileri kullanırlar. Bilgi girişleri argüman listesi olarak verilir. Argüman listesinin ön tanımlı değerleri de olabilir. Önce basit bir fonksiyondan başlayalım. Yazacağımız bu fonksiyona sadece bir tane argüman verelim ve verdiğimiz argümanı ekrana yazdıralım:

program6_2.py

```
#!c:\\python22\\python.exe

def ikinciFonksiyon(gelen):
        print gelen

ikinciFonksiyon("Mustafa BAŞER")
ikinciFonksiyon(3*4)
```

Yukarıdaki örnekte **ikinciFonksiyon** bizden bir tane argüman istiyor. Diğer dillerin tersine argümanın hangi cinsten olduğunu belirtmiyoruz. Fonksiyonu ilk çağırışımızda bir cümle değer gönderdik, diğerinde ise bir tamsayı gönderdik. Başka bir seferinde liste, sözlük veya diğer Python nesnelerinden birisini gönderebilirdik. Bu programı çalıştırırsak örnek çıktı şu şekilde olmalıdır:

```
Mustafa BAŞER
12
```

Eğer siz bir fonksiyona sağlanan verilerin belirli cinsten olmasını istiyorsanız bunu kendiniz denetlemelisiniz. Aksi taktirde Python hata kodu verecektir.

Fonksiyonlara sağlanan argümanlar aralarına virgül konularak ayrılır. Gönderilen argüman sayısı ile istenilen argüman sayısı aynı olmalıdır. Şimdide gönderdiğimiz iki değerin çarpımını ekrana yazan başka bir basit fonksiyon yazalım:

program6_3.py

```
#!c:\\python22\\python.exe

def carpim(x,y):
        print x*y

carpim(9,4)
carpim("mbaser",3)
```

Bu fonksiyona sağlanan değerler yine herhangi bir Python veri tipi olabiliyor. Fonksiyonu ilk çağırdığımızda iki tamsayının çarpımını alıyoruz. Daha önceden de öğrendiğimiz gibi bir tamsayı ile bir cümle çarpılabilir. Bu programı yazıp çalıştırırsanız çıktısı şu şekilde olacaktır:

```
36
mbasermbasermbaser
```

Eğer **carpim()** fonksiyonunu çağırırken yeteri kadar argüman vermezsek (veya daha fazla) bir hata ile karşılaşırız. Örneğin **carpim(9)** şeklinde bir çağrım yaparsanız:

```
Traceback (innermost last):
Traceback (most recent call last):
  File "C:/windows/Desktop/Python Kitabı/program6-3.py", line 6,
in ?
    carpim(9)
TypeError: carpim() takes exactly 2 arguments (1 given)
```

gibi bir hata uyarısı alırsınız. Yani fonksiyon iki argüman istiyor, siz bir tane vermişsiniz anlamında. Bazen yazdığımız fonksiyonun yeteri kadar argüman verilmediğinde de çalışmasını isteyebiliriz. Bu gibi durumlarda bir ön tanımlı değer verilmelidir. Bunu hemen argüman listesinde belirleyebilirsiniz. Şimdi böyle bir fonksiyon yazalım:

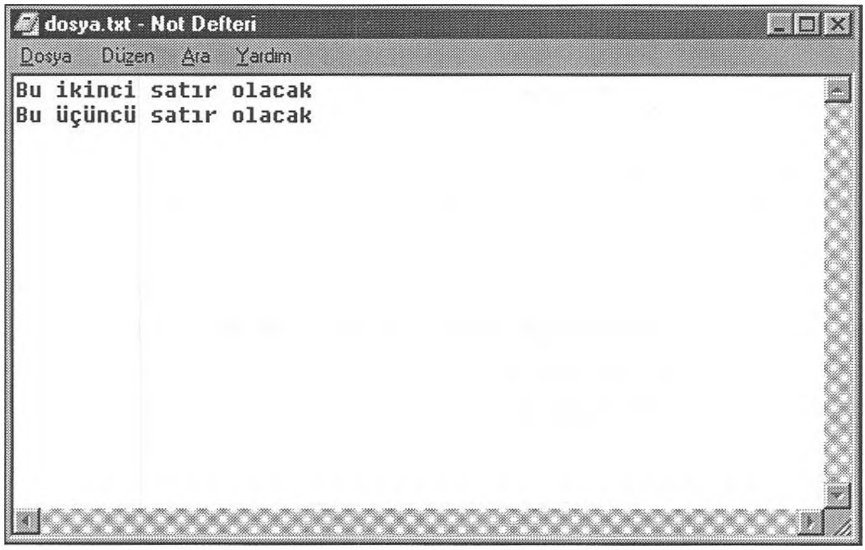
program6_4.py

```
#!c:\\python22\\python.exe

def dosyayaYaz(metin,mod='w'):
        d=open('c:\dosya.txt',mod)
        d.write(metin+'\n')
        d.close()

dosyayaYaz('Bu ilk satır olacak')
dosyayaYaz('Bu ikinci satır olacak')
dosyayaYaz('Bu üçüncü satır olacak','a')
```

Bu programı çalıştırdığımızda **dosya.txt** dosyasının içeriği **Şekil-6.1**'deki gibi olacaktır. Birinci satırın neden kaybolduğunu açıklayınız.



Şekil-6.1: Birinci satırımız yok olmuş. Neden?

Bu programda tanımladığımız **dosyayaYaz()** fonksiyonu iki tane argüman alabilir. İlk argüman dosyaya yazılacak olan cümle ikincisi ise dosyanın hangi mod ile açılacağıdır. Eğer dosya yazma modu belirtilmezse, dosya "w" ile açılacak ve önceki bilgiler kaybolacaktır. Fonksiyonun ilk iki cağırılışında mod belirtilmediğinden dosya yeniden açıldı. Son çağırılışta dosya modu "a" (yani sonuna ekleme) argümanı fonksiyona gönderildiğinden, dosyanın sonuna ekleme yapıldı.

Bilgi Girişi Olmayıp Geriye Bilgi Döndüren Fonksiyonlar

Bazen bilgi vermediğimiz halde geriye bilgi döndüren bir fonksiyona ihtiyacımız olabilir. Örneğin zamanı Türkçe olarak veren bir fonksiyon yazmak istersek bu fonksiyona herhangi bir şekilde parametre vermeden geriye bir bilgi dönüşü olacaktır. Fonksiyonlar bilgileri geriye **return** hazır fonksiyonu ile döndürürler.

program6_5.py

```
#!c:\\python22\\python.exe

import locale, time
locale.setlocale(locale.LC_ALL,"Turkish_Turkey.1254")

def zaman():
        TurkceZaman=time.strftime("%a %b %d %H:%M:%S %Y",
time.lo caltime(time.time()))
         return TurkceZaman

print zaman()
```

Bu programı yazıp çalıştırdığınızda şu çıktıyı alacaksınız:

```
Pzt Ara 31 15:02:12 2001
```

Gördüğünüz gibi sonuç tam anlamıyla harika. Fonksiyon zamanı Türkçe olarak verdi. Önce fonksiyonun nasıl işlediğine bir bakalım, daha sonra diğer ayrıntıları tartışırız.

Bu fonksiyona herhangi bir şekilde bilgi giriş sağlamıyoruz, ancak sonuçta bize bilgi dönüyor. Bilgi bilgisayarınızın iç saatinden alınan zamanın işlenerek geriye döndürülmesinden ibaret. Bir fonksiyon ile geriye bilgi dönüşü olacaksa, döndürülen bilgi **return** ön tanımlı fonksiyonu tarafından yaptırılır. **return** birden çok parametre de alabilir. Eğer birden çok parametre verilmişse, geriye dönen bilgi tüp şeklindedir. Hemen basit bir örnek verelim:

```
>>> def x():
...      return 9,5,3
...
>>>x()
(9, 5, 3)
```

Burada gördüğünüz üzere basit bir **x** fonksiyonu geriye üç tane bilgi gönderiyor. Aslında bu bilgi tüp olarak döndüğünden tek bir bilgi gibi sayılabilir.

x() fonksiyonunun geri dönüşümü şu şekilde değişkenlere aktarılabilir:

```
>>> (ilk, ikinci, ucuncu)=x()
>>> ilk
9
>>> ikinci
5
>>> ucuncu
3
```

Ya da geri dönüş bir tüp şeklinde olduğundan indisler yazılarak bilgiler kullanılabilir. Bunu şu şekilde örnekleyebiliriz:

```
>>> x()[0]
9
>>> x()[1]
5
>>> x()[2]
3
```

Bu çağrımda fonksiyonun her seferinde yeniden işletildiğini unutmayın.

Aslında bu şekildeki fonksiyonları daha kısa bir şekilde kullanabilirsiniz. Bunu fonksiyonların atanması konusunda göreceğiz.

return'ün geriye bilgi döndürdüğünü biliyoruz. Eğer fonksiyon herhangi bir nedenle **return** fonksiyonuna rastlarsa geri kalan kısımlar işletilmez. Örneğin aşağıdaki fonksiyonda **if**'den sonra **else** kullanmanıza gerek yok, çünkü **if** doğrulandığında hemen işlem kesilir ve geri kalan kod işletilmez. Doğrulanmazsa sonraki satırlar işletilir:

```
>>> def deneme(gelen):
                if gelen < 5: return "Gelen sayı beşten küçük"
                return "Gelen sayı beşten büyük"

>>> print deneme(4)
Gelen sayı beşten küçük
>>> print deneme(9)

Gelen sayı beşten büyük
```

Şimdi asıl programımıza geri dönelim ve olup bitenleri anlamaya çalışalım:

(import) satırında iki tane modülü yüklüyoruz. Bunlar **locale** (yerel) ve **time** (zaman). Modülleri ilerde anlatacağımızdan şimdilik bu kadarını bilmeniz yeterli. Hemen bir altındaki satırda, zamanı Türkçe alabilmek için **locale** modülünde bulunan **setlocale** hazır fonksiyonu ile yerelimizi Türçe yapıyoruz. Eğer UN*X kullanıyorsanız, çevre değişkenlerinizde **LANG** halihazırda 'tr_TR' şeklindeyse bu işlemleri yapmadan da sonucu Türkçe olarak alabilirsiniz. Ancak bazı programları çok dilli yazmak isteyebilirsiniz. ve işte o zaman yerele program içerisinden müdahele etme şansınız olacaktır. Özellikle çok dilli web uygulamaları geliştirmeyi düşünüyorsanız...

(time) modülünde bulunana **time()** hazır fonksiyonu bilgisayarınızın iç saatinde bulunan zamanı 1970 yılından itibaren saniye cinsinden verecektir. Şimdi Python kabuğunda bunu görelim:

```
>>> import time
zamanSaf=time.time()
>>> zamanSaf
1009804118.55
```

Gördüğünüz gibi sonuç bizim için pek bir anlam taşımıyor. Python zamanın saliselerini ondalık sayı olarak verir.

(time) modülü içerisinde bulunan **localtime()** hazır fonksiyonu, saniye cinsinden verilen argümanı bizim anlayacağımız bir dile çevirir. Şimdi zamanSaf değerini **localtime()** fonksiyonuna verelim:

```
>>> yerelZaman=time.localtime(zamanSaf)
>>> yerelZaman
(2001, 12, 31, 15, 8, 38, 0, 365, 0)
```

Gördüğünüz gibi bu fonksiyon bize 9 elemanlı bir tüp verdi. Şimdi bu elemanların ne anlama geldiğini sıralıyalım:

Zaman Elemanı	Anlamı
yerelZaman[0]	yıl
yerelZaman[1]	ay
yerelZaman[2]	gün
yerelZaman[3]	saat
yerelZaman[4]	dakika
yerelZaman[5]	saniye
yerelZaman[6]	haftanın kaçıncı günü olduğu
yerelZaman[7]	yılın kaçıncı günü olduğu
yerelZaman[8]	gün ışığı kaydetme anahtarı

strftime() fonksiyonu ise, bu verileri anlaşılır bir dile çevirir. Şimdi bu fonksiyon tarafından kullanılan bazı yönergelere bir göz atalım:

Yönerge	Anlamı
%a	hafta gününün kısaltılmış ve yerelleştirilmişi (Paz, Sal...)
%A	hafta gününün yerelleştirilmişi (Pazartesi, Salı...)
%b	ayın kısaltılmış ve yerelleştirilmişi (Oca, Şub...)
%B	ayın yerelleştirilmişi (Ocak, Şubat,..)
%d	ayın kaçıncı günü olduğu
%H	24'lük sistemde saat
%I	12'lik sistemde saat
%m	kaçıncı ay olduğu
%p	yerelleştirilmiş olarak öğleden önce veya sonra (ÖÖ, ÖS)
%s	saniye
%w	haftanın kaçıncı günü olduğu
%y	yüzyıl olmaksızın yıl (99, 00, 01...)
%Y	yüzyıl ile birlikte yıl (1999, 2000, 2001...)

Elbetteki time modülünün kullanılması bu anlattıklarımızla kısıtlı değildir. Bir dizin ya da dosyanın oluşturulduğu tarihi de böyle anlaşılabilir bir cinse getirmeniz mümkün.

Bu modülde bulunan ve işinize yarayacak diğer bir fonksiyon da sleep() tir. Bu fonksiyon program akışını belirlediğiniz sürece durdurur. Verilen süre saniye cinsindendir. Örneğin:

```
>>> time.sleep(10)
```

yazarsanız, python 10 saniye sürece kendisini uyutacaktır.

Bir Bilgi Giriş Olup Bilgi Döndürmeyen Fonksiyonlar

Bu tip fonksiyonlara argüman olarak bilgi (ya da bilgiler) verdiğimiz halde geriye bilgi döndürmezler. Fonksiyona bilgiler aktarılırken arasına virgül konur. İsterseniz bilgi girişini tek tutup, bilgiler kümesini bir liste içerisinde de gönderebilirsiniz. Ancak bu yol pek tercih edilmez. Hemen kısa bir örnek verelim:

```
>>> def verileniYaz(gelenBilgi):
...     print "Bu fonksiyona gönderilen bilgi: ", gelenBilgi
...
>>> verileniYaz('Bir cümle gönderiyorum')

Bu fonksiyona gönderilen bilgi: Bir cümle gönderiyorum

>>> verileniYaz(['Bunlar', 'liste','elemanlarıdır'])

Bu fonksiyona gönderilen bilgi:['Bunlar', 'liste', 'elemanlar\375d\375r']
```

Yukarıda basit bir fonksiyonun tanımlanmasını gördünüz. Bir fonksiyona her zaman istediği kadar bilgi göndermek zorunda değilsiniz. Bazı durumlarda gönderilmeyen argümanın ön tanımlı değerlerinin alınması gerekir. Örneğin kullanıcı adını ve alanı birleştirip e-posta adresine çeviren bir fonksiyon yazalım. Eğer alan verilmezse kullanıcı kolaymail.com üyesi kabul edilsin. Bu durumda:

```
>>> def ePostaYap(hesap, alan='kolaymail.com'):
...     print '%s@%s' % (hesap, alan)
...
```

```
>>> ePostaYap('mbaser')
mbaser@kolaymail.com
>>> ePostaYap('akgun','gelecek.com.tr')
akgun@gelecek.com.tr
```

şeklinde bir yazılım gerçekleştirilebilir. **ePostaYap()** fonksiyonu ilk çağrıldığında sadece kullanıcı adı verildi, alan verilmediğinden otomatik olarak ön tanımlı değer alındı. İkinci kez çağrıldığında ise alan da verilmiş oldu ve ön tanımlı değer göz ardı edildi.

Eğer bir fonksiyonun argümanlarından birinin veya birkaçının ön tanımlı değeri olacaksa, bunlar en sona yazılmalıdır, yani ön tanımlı değeri olmayan argümanlar öncelikle yazılmalıdır.



Yukarıda gördüğünüz gibi, fonksiyon tarafından alınan değerler ile gönderilen değerler aynı sırada veriliyor. Python o kadar esnektir ki, bu sırayı da değiştirmenize olanak tanır. Eğer fonksiyona gönderilen argümanları istenilen sırada vermek istemiyorsanız, gönderirken alınacak olan değerin ne olduğunu belirtebilirsiniz. Şimdi bu bilgiler ışığında yukarıdaki fonksiyonu çağıralım:

```
>>> ePostaYap(alan='harmony.com.tr', hesap='falkan')
falkan@harmony.com.tr
```

Python'un esnekliği bu kadarla kalmıyor. Fonksiyona sağlanan argümanların bir tüpe alınmasını sağlayabilirsiniz. Şimdi **ePostaYap()** fonksiyonunu bu türe çevirelim:

```
>>> def ePostaYap(*gelenler):
...     return gelenler[0]+'@'+gelenler[1]
...
>>> ePostaYap('akgun','gelecek.com.tr')
'akgun@gelecek.com.tr'
```

Burada **ePostaYap()** fonksiyonuna sağladığımız argümanlar gelenler adlı bir tüpe aktarıldı. Daha sonra bu tüpün elemanları kullanıldı. Yukarıdaki

çağrımda gelenler tüpü ('akgun', 'gelecek.com.tr') şeklinde oldu. Buradaki yıldızın, C programlama dilindeki pointer'larla bir alakası olmadığını belirtelim.

Şimdi başka bir örnek verelim:

```
>>> def denemeFonksiyonu(ilkArguman, *digerleri):
...     print ilkArguman
...     print digerleri
...
>>> denemeFonksiyonu('ilk bilgi', 'ikinci bilgi', 'ucuncu bilgi')
ilk bilgi
('ikinci bilgi', 'ucuncu bilgi')
```

Bu tanımlamada, gönderilen argümanların ilki ile diğerlerini ayırmış olduk. Fonksiyonlara bilgi girişi yaparken, bilgilerin ne türden bir değer olduğu kuşkusuz çok önemlidir. Örneğin, eğer gönderilen argümanlar bir bölme işleminde kullanılacaksa, tamsayı ya da ondalık sayılardan başka bir argüman gönderilmemelidir. Hemen bunu örnekleyelim:

```
>>> def bolme(a,b):
...     print a/b
...
>>> bolme(9,3)
3
```

Bu kadar kısa bir fonksiyonu çağırırken hata yapmayabilirsiniz, ancak karmaşık ve de çok önceden yazılan fonksiyonları kullanıyorsanız ya da sizin yazdığınız fonksiyonları başkaları kullanıyorsa, fonksiyona gelen bilgilerin istenilen türden olup olmadığı denetlenmelidir. Şimdi bu fonksiyonu yanlışlıkla şu şekilde çağıralım:

```
>>> bolme(9,'elma')
Traceback (most recent call last):
  File "<pyshell#43>", line 1, in ?
    bolme(9,'elma')
```

```
  File "<pyshell#42>", line 2, in bolme
    print a/b

TypeError: unsupported operand type(s) for /
```

Bu gibi durumlarda hata mesajı verilecektir. Bunun yerine çağıranı uyarmak daha faydalı olacaktır. Bu amaçla **type()** ön tanımlı fonksiyonu kullanılır. Bu fonksiyon, bilgilerin hangi türden olduğunu belirtir:

```
>>> type(98)
<type 'int'>
>>> type('mbaser')
<type 'string'>
>>> type([5,6,'fatih'])
<type 'list'>
```

Yukarıda sıra ile, int (tamsayı), string (cümle), list (liste) argümanları verdik ve türlerini denetledik. Şimdi bölme fonksiyonuna gönderilen argümanların ondalık (real) veya tamsayı (int) olma durumunu test etmek istersek:

```
>>> def bolme(a,b):
...       if (type(a)==type(1) or type(a)==type(1.2)) and
(type(b)==type(1) or type(b)==type(1.2)):
...             return a/b
...       else:
...           return ( a, 've', b, 'birbirlerine bölünemezler')
...
>>> bolme(96,9)
10
>>> bolme(96,9.3)
10.3225806452
>>> bolme(96.3,'v')
(96.3, 've', 'v', 'birbirlerine b\366l\374nemezler')
```

Bilgi Girişi ve Dönüşü Olan Fonksiyonlar

Şimdiye kadarki açıklamalarımızla bunların yazılması ve çağrılmasının nasıl olabileceğini tahmin ediyor olmalısınız. Şimdi yukarıdaki fonksiyonu bu türden bir fonksiyon haline getirelim:

```
>>> def ePostaYap(hesap, alan='kolaymail.com'):
...     return '%s@%s' % (hesap, alan)
...
>>> benimPostaAdresim=ePostaYap('mbaser')
>>> benimPostaAdresim
'mbaser@kolaymail.com'
```

Bilgi girişi ve dönüşü için yukarıda anlattığımız açıklamalar geçerlidir.

Menüler

Etkileşimli programcılığın en önemli unsurlarından birisi de menülerdir. Kullanıcının sağladığı bilgiler doğrultusunda, program akışı değiştirilir veya ilk sağlanılan veri doğrultusunda programın belirli bir fonksiyonunun işletilmesi sağlanabilir. Bunu açıklamak için sadece ve sadece eğitim amaçlı ve bir işe yaramayan şöyle bir program yazalım:

program6_6.py

```
#!c:\\python22\\python.exe

def carpim():
    print "Bu fonksiyon çarpma işlemini yapacak"

def toplam():
    print "Bu fonksiyon toplama işlemini yapacak"

def bolum():
    print "Bu fonksiyon bölme işlemini yapacak"

istenen=''

while istenen <> 'E':
```

```
print
print "Hangi işlemin yapılmasını istiyorsunuz?"
print "[Ç]arpım [T]oplam [B]ölme [E]xit"

istenen=raw_input()

if istenen=='Ç': carpim()
elif istenen=='T': toplam()
elif istenen=='B': bolum()
elif istenen=='E': print "Şimdi Çıkılıyor"
else: print "Geçersiz işlem"
```

Programı çalıştırırsak:

```
Hangi işlemin yapılmasını istiyorsunuz?
[Ç]arpım [T]oplam [B]ölme [E]xit
Ç
Bu fonksiyon çarpma işlemini yapacak

Hangi işlemin yapılmasını istiyorsunuz?
[Ç]arpım [T]oplam [B]ölme [E]xit
T
Bu fonksiyon toplama işlemini yapacak

Hangi işlemin yapılmasını istiyorsunuz?
[Ç]arpım [T]oplam [B]ölme [E]xit
E
Şimdi Çıkılıyor
```

Evet klasik bir programcı menüleri aynen böyle oluşturur. Ancak bu Python mantığına oldukça ters bir düşünce. Her eklenen menü için yeni bir karşılaştırma yapılması gerekiyor ve kod hem karmaşıklaşıyor hem de uzuyor. Eğer Python'da **case** ifadesi olsaydı belki biraz daha kolay olacaktı, ancak öyle bir ifade yok. Python'ca düşünelim: "Her şey basit ve anlaşılır olurken kod ise kısacık olmalı." Bu düşünceyi menümüze uygulamaya çalışalım. Öncelikle Python'da her şeyin bir nesne olduğunu hatırlayalım. Fonksiyon-

lar'da birer nesnedirler. Sözlükler ise nesnelerin bir araya gelmesinden oluşur. Evet, ben fonksiyonlarımı bir sözlük içerisine koyabilirim. Kullanıcının verdiklerini anahtar, değer olarak fonksiyonları bunlara karşı yapılacak işler olarak belirlerim. İşte bu kadar. O halde ismi **menu** olan şöyle bir sözlük tasarlayabilirm:

```
menu={
        'C':carpim,
        'B': bolme,
        'T': toplama,
        'E':cikis
}
```

Şimdi bunu programımız içerisine koyalım:

program6_7.py

```
#!c:\\python22\\python.exe

def carpim():
    print "Bu fonksiyon çarpma işlemini yapacak"

def toplam():
    print "Bu fonksiyon toplama işlemini yapacak"

def bolum():
    print "Bu fonksiyon bölme işlemini yapacak"

def cikis():
    print "Şimdi çıkılacak"


istenen=''

while istenen <> 'E':
```

```
print
print "Hangi işlemin yapılmasını istiyorsunuz?"
print "[C]arpım [T]oplam [B]ölme [E]xit"

istenen=raw_input()

menu={
          'C':carpim,
          'B': bolum,
          'T': toplam,
          'E':cikis
}

menu[istenen]()
```

Şimdide bu programı çalıştırıp sonuca bakalım:

```
Hangi işlemin yapılmasını istiyorsunuz?
[C]arpım [T]oplam [B]ölme [E]xit
C
Bu fonksiyon çarpma işlemini yapacak

Hangi işlemin yapılmasını istiyorsunuz?
[C]arpım [T]oplam [B]ölme [E]xit
T
Bu fonksiyon toplama işlemini yapacak

Hangi işlemin yapılmasını istiyorsunuz?
[C]arpım [T]oplam [B]ölme [E]xit
B
Bu fonksiyon bölme işlemini yapacak

Hangi işlemin yapılmasını istiyorsunuz?
[C]arpım [T]oplam [B]ölme [E]xit
E
Şimdi çıkılacak
```

Özellikle CGI programcılığında kullanıcın, hangi düğmeye tıkladığını belirleyerek gidilecek yolun karar verilmesinde bu tür menüler oldukça sık kullanılmaktadır. Kullanıcı tarayıcılarına gönderilen çerezlere (cookie) göre ona özel web sayfalarının tasarlanmasında da elverişli bir menü tanımlama yöntemidir. Aynı işi **if-elif**'lerle yapacaksanız neden Python öğreniyorsunuz ki? Bu dilin kolaylıklarından doyumsuzca yararlanın ve keyfinize bakın.

Bu tür menüler listeler ile de yapılabilir. Ancak en çok tercih edileni sözlüklerdir. Python sen bir harikasın!

Fonksiyonlarla İlgili Hazır Fonksiyonlar

Fonksiyonlarla çalışırken, Python size olağanüstü kolaylıklar sunar. Bu kolaylıklardan yararlanmak hem programınızın performansını artırır hem de yazılımı oldukça kısaltır. İyi bir programcı aşağıdaki hazır fonksiyonları kullanmak için fırsat kollar.

map

map() hazır fonksiyonu, kendisine verilen fonksiyon ile, bu fonksiyona gönderilecek olan bir dizi değeri gönderir ve sonucu bir listeye atar.

Programlarınızı yazarken, bazen bir fonksiyonu belirli bir liste için teker teker çağırmak durumunda kalırsınız. Örneğin kullanıcının programa argüman olarak verdiği değerler her zaman için cümle (string) tipindedir. Siz bunları rakam olarak kullanmak isterseniz **int()** fonksiyonu ile teker teker sayı değerine çevirmek durumundasınız. Hemen bir örnek verelim:

program6_8.py

```
#!c:\\python22\\python.exe

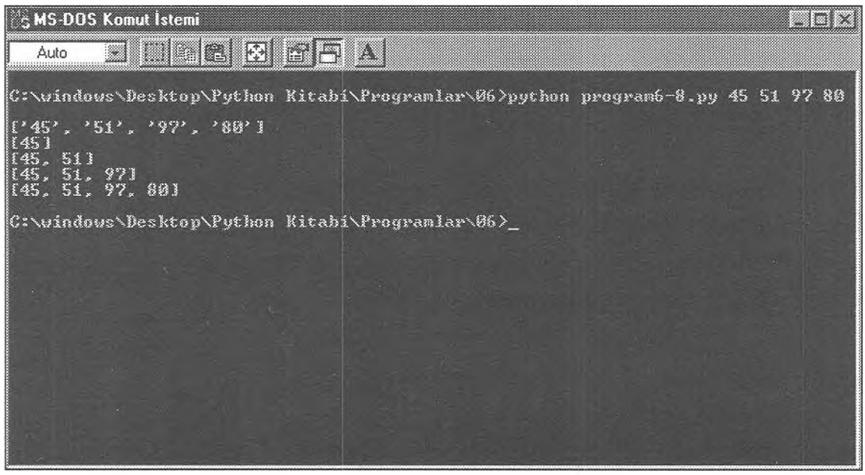
import sys

sayiDeger=[]

print sys.argv[1:len(sys.argv)]

for k in sys.argv[1:len(sys.argv)]:
    sayiDeger.append(int(k))
    print sayiDeger
```

Bu programı **program6_8.py** ismi ile kaydedip **45 51 97 80** argümanları ile çalıştırırsak **Şekil-6.2**'de görünen sonucu elde ederiz:



Şekil-6.2: Python programının komut satırından argümanlarla çalıştırılması

Gördüğünüz gibi, argüman listesi cümle tipinden olurken biz bunları sayı değerlerle değiştirdik. Artık bu değerlerle aritmetik işlemler yapılabilir. Ancak **map()** hazır fonksiyonu varken döngünün yapılmasına gerek yoktur. Yukarıdaki işlemi kısaca şu şekilde yazabiliriz:

program6_9.py

```
#!c:\\python22\\python.exe

import sys

print sys.argv[1:len(sys.argv)]
sayiDeger=map(int,sys.argv[1:len(sys.argv)])

print sayiDeger
```

Programı çalıştıracak olursak:

```
C:\windows\Desktop\Python Kitabı>python program6-9.py 45 51 97 80
['45', '51', '97', '80']
[45, 51, 97, 80]
```

Bu şekildeki bir yazılım hem kodu kısaltacak hem de programın anlaşılırlığını artıracaktır. Unutmayın, Python çok mantıklı ve açık bir yazılıma sahiptir. Sizi her zaman basit düşünmeye ve temiz bir kod yazmaya yönlendirir. Birçok C programcısı Python'u tanıdıktan sonra daha temiz, daha anlaşılır ve daha basit kodlar yazabildiklerini defalarca itiraf etmişlerdir. Çok karmaşık gibi görünen şeyleri çok basit kodlarla halledebilirsiniz.

Bu yazılımın kodu bir öncekine göre bayağı kısadır. Bir defa **sayiDeger** gibi bir boş listenin tanımlanmasına gerek kalmıyor, öte yandan bir döngü yapmak zorunda değilsiniz. Son olarak **append()** ile daha önce tanımladığınız listeye ekleme yapmak durumunda değilsiniz.

map() fonksiyonun yazılımını şöyle özetleyebiliriz:

map(*<fonksiyon>, <fonksiyona_gidecek_değerlerin_listesi>*)

lambda()

Bazen o kadar basit fonksiyonlar tanımlarız ki, bunun **def** ile yapılması kodu uzatabilir. Daha basit ve temiz bir yazılım arzusundaysanız bu gibi durumlarda **lambda** sizin işinizi görecektir. Örneğin iki sayının çarpımını geriye dündüren bir fonksiyon yazmak istesek, klasik bir programcı mantığıyla şu kodu yazarız:

```
>>> def carp(x,y):
...     return x*y
...
>>> carp(2,5)
10
```

Bunun yerine daha modern bir yaklaşım izlersek, **lambda()** fonksiyonunu kullanırız:

```
>>> a=lambda x,y : x*y
>>> a(2,5)
10
```

Gördüğünüz gibi kod oldukça kısalıyor. **lambda** sadece basit bir ifadeyle işlem yapan kısa fonksiyonların tanımlanması için kullanılabilir. Yazılımı şöyledir:

lambda *<parametre_listesi>: <ifade>*

Eğer fonksiyonun parametrelerinin ön tanımlı değerlerinin olmasını istiyorsanız, normal bir fonksiyonlarda yaptığınız işlemleri yapabilirsiniz:

```
>>> b=lambda x=5,y=4 : x*y
>>> b()
20
>>> b(3)
12
>>> b(3,9)
27
```

İsim Alanları

İsim alanları ile ilgili daha ayrıntılı bilgiyi sınıflar bölümünde göreceğiz, şimdilik yeteri kadar açıklama verelim.

Python'da bir değişkene aksi belirtilmedikçe yerel alandan erişilebilir. Bu alan dışında aynı isimde başka değişkenler/nesneler tanımlanabilir. Şu örneği inceleyelim:

```
>>> a=99
>>> def f():
          a=191
          print "f() fonksiyonunda a değişkeninin değeri:",a

>>> print a
99
>>> f()
f() fonksiyonunda a değişkeninin değeri: 191
>>> print a
99
```

A değişkenini en başta tanımladık. Daha sonra **f()** fonksiyonu içerisinde aynı isimde başka bir değişken tanımlamış olduk. Bu fonksiyonu çağırdığımız halde **a** değişkenini fonksiyon dışından tekrar çağırdığımızda değeri yine 99 oldu. Şimdi de şunu deneyelim:

```
>>> a=99
>>> def f():
            a=a+200
            print "f() fonksiyonunda a değişkeninin değeri:",a
>>> f()
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in ?
    f()
  File "<pyshell#22>", line 2, in f
    a=a+200
UnboundLocalError: local variable 'a' referenced before assign-
ment
```

Gördüğünüz gibi **a** değişkeni başta tanımlandığı halde fonksiyon içerisinden buna erişilemedi. Nesneleri program akışı içerisinde her yerden görmek için **global** ön tanımlı fonksiyonu kullanılır.

```
>>> a=99
>>> def f():
            global a
            a=a+200
            print "f() fonksiyonunda a değişkeninin değeri:",a
>>> f()
f() fonksiyonunda a değişkeninin değeri: 299
>>> print a
299
```

Fonksiyon Açıklamaları

Yazdığınız programı ister sadece siz, isterse başkaları kullansın, tanımladığınız fonksiyonlarla ilgili açıklama satırları yazmazsanız ileride bu fonksiyonu kullanırken tekrar incelemek, ne işe yaradığını görmek ve nasıl kullanıldığını anlamak için gereksiz yere zaman kaybedersiniz. İyi yazılmış kodlarda za-

man zaman açıklamalar bulunur. Özellikle fonksiyonların başında ne işe yaradığı, hangi argümanları aldığı ve sonuçta nasıl bir bilgi döndürdüğü ile ilgili açıklamalar yazmalıdır. Klasik bir programlama dilinde bunlar açıklama satırı olarak barındırılır, fonksiyonu sonradan çağıracak kişi gidip kaynak kodu ya da bir **man** sayfasını okumak durumundadır. Python'da ise her kolaylık düşünülmüş ve bu açıklamaların istenildiğinde çağırılmasına imkan tanınmıştır. İlerki bölümlerde dökümantasyonla ilgili geniş açıklamalar bulacaksınız. Burada sadece fonksiyonlarla ilgili açıklamaların nasıl yazılıp çağırıldığına bakacağız.

Geçtiğimizi bölümlerde **time.time()** fonksiyonunu kullandık. Peki bu fonksiyonla ilgili bir açıklama isteseydik? Bunu **__doc__** özel çağırımı ile yapabiliriz:

```
>>> import time
>>> print time.time.__doc__
time() -> floating point number

Return the current time in seconds since the Epoch.
Fractions of a second may be present if the system clock provi-
des them.
>>>
```

Gördüğünüz gibi **time.time()** fonksiyonunun kullanımı ile ilgili kısa bir açıklamaya eriştik. Bu gibi açıklamaları kendi fonksiyonlarınızda, fonksiyonu tanımlar tanımlamaz hemen altına tırnak içinde yazabilirsiniz. Eğer açıklama uzun olacaksa, üç tırnak da kulanabilirsiniz. Örneğin **carpim** fonksiyonu ile ilgili bir açıklama yazalım:

```
>>> def carpim(x,y):
...     'Varilen sayıların çarpımını döndüren bir fonksiyondur.'
...     return x*y
...
>>> print carpim.__doc__
Varilen sayıların .çarpımını döndüren bir fonksiyondur.
>>> carpim(5,6)
30
```

Python Modülleri

07

Şimdiye kadar Python modüllerini mecburen kullanmak zorunda kaldık. Python modüler bir yapıda çalıştığından bundan kaçış yoktur. İyi bir programcı kodlarını modüler olarak yazar. Modüler ne demek? Programı parçaları ayırıp, bu parçaları gerektiği zaman kullanmak demektir. Örneğin bir dizi CGI programı yazacaksınız. Her CGI programında kaçınılmaz unsurlar vardır. Bunların en başında kullanıcı onayı gelir. Eğer kullanıcı onayı işini bir modüle (aslında modülden kastımız bir Python dosyası) yazarsanız, her programda bu modülü çağırır, kullanıcının giriş yapmış veya yapmamış olduğunu denetlersiniz. Bu tür programlarda bir diğer konu da oturum takibidir. Her oturumda kullanıcıların çeşitli değişkenler için aldıkları çeşitli değerler vardır. Ya bu değerleri her seferinde kullanıcıya gönderir tekrar istersiniz ya da yerel diskte o kullanıcının oturum numarası altında toplarsınız ve bu şekilde hem zamandan kazanır hem de daha güvenilir bir iş yapmış olursunuz. Bunu her programa eklemek yerine bir modül hale getirir ve işinizi oldukça kolaylaştırırsınız.

Neden Modüler Programcılık?

Programlarınızı modüler yazmanın bir çok avantajı vardır. Bunlardan ilki yukarıda bahsettiğimiz, kodun tekrar tekrar kullanılmasıdır. İkincisi ise program başlangıcının hızlanmasıdır. Eğer bir kodu modüler olarak yazar ve çağırırsanız, kod hemen byte-code denilen bir yapıya derlenir ve bu Python yorumlayıcısı tarafından yorumlanır. Modülün kaynağında herhangi bir değişiklik yapılmamışsa derleme işlemi yapılmadan direk byte-code kullanılır. Hatta kaynağı silerseniz, kaynakta değişiklik var mı yok mu diye bakılmadan doğrudan derlenmiş olan dosya yorumlanır. Böylelikle programınızın başlatılması inanılmaz derecede hızlanmış olur. Ancak şunu unutmamak gerekir ki, programın kaynaktan ya da byte-kod'dan yorumlanması onun hızını etkilemez, sadece program başlangıcı hızlanır. Zaten program bir kere başlatıldı mı gerisi tatminkar bir hızda işler. Modüler yazımın üçüncü bir avanta-

ı da, yazılan kodda bir değişiklik yapmak gerektiğinde, o modülü kullanan tüm programların değişen koda göre işlemesidir. Düşünün aynı fonksiyonu on beş program tarafından kullanıyorsunuz, eğer aynı kodu tüm beş programlar içerisine kopyala-yapıştır şeklinde yapmışsanız, fonksiyonda gerekli bir değişikliğin hepsinde yapılması gerekir.

Bir Python modülü, Python komut ve deyimlerinden meydana gelir. Aslında yazılan her Python programı aynı zamanda bir modüldür. Ancak kodun modül olarak mı, program olarak mı çalıştırıldığı özel bir fonksiyon ile denetlenebilir. Bunu aşağıda göreceksiniz. Python modülleri her zaman Python kodlarından oluşmayabilir. Özellikle performansın artırılması için ve sistemde bulunan diğer hazır kütüphanelerin kullanılması açısından bunlar her zaman Python kodlarından oluşmayabilir. Bu tür modüller genelde C (veya C++, Delphi) ile yazılır. Ancak yazılan bu modül (başka bir diğer deyişle kütüphane) sadece Python tarafından kullanılabilecek şekilde tasarlanır. Python içerisinde C kütüphanelerini de çağırabilirsiniz. Ancak bu ileri C ve Python'un derinlemesine bilinmesi ile gerçekleşir. Zaten hiçbir zaman böyle bir şeye ihtiyacınız olmayacaktır, çünkü Python o kadar hızlı bir şekilde geliştiriliyor ki, teknolojik yenilikler yapılıp, kullanıcılara en hızlı bir şekilde aktarılıyor. Bu bakımdan hiçbir kuşkunuz olmasın. Ancak temel programlama komutlarında herhangi bir değişiklik yapılmıyor, Python'un ilk sürümlerinde yazılan kodlar bile hala çalıştırılabilir durumda. Python yazarı Python'u geliştirken geriye uyumluluğun yüzde yüz uyumlu çok dikkat etmektedir.



Aşağıda vereceğimiz tüm örnek modüller C:\\python22 klasörüne kaydedilecektir. Eğer bu klasöre yazma hakkınız yoksa, önce bölüm 7.7'yi okuyarak, modülleri kaydedeceğiniz klasörün tam patikasını Python yorumlayıcısını her çağırdığınızda eklemeniz gerekiyor. UN*X kullanıcılarının modülleri bulundukları klasöre kaydedip, Python yorumlayıcısını oradan çalıştırmaktan başka herhangi birşey yapmalarına gerek yoktur.

Modüllerin Yazılması ve Çağrılması

Python modülleri **import** ön tanımlı komutu ile programa dahil edilir. Zaten daha önceden birçok hazır Python modülünü çağırdınız. Şimdi hemen bir **Python** modül yazıp bunu çağıralım. Aşağıdaki kodları yazıp **"c:\python22"** klasörüne ismiyle kaydedin:

modul7_1.py

```
a=101
isim='Mustafa BAŞER'
```

Önce nelerin ön tanımlı olarak kullanılabilir olduğunu görmek için **dir()** yazalım ve sonra aşağıdaki gibi "**import modul7-1**" diyelim.

```
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__']
>>> import modul7_1
```

dir() iç tanımlı fonksiyonu, o andaki isim alanımızda erişilebilir nesnelerden oluşan bir liste verir. İsim uzayları ile ilgili geniş açıklamayı sınıflar bölümünde göreceksiniz.

Yazdığımız modülü **import** ile eklediğiniz halde hiçbir şey olmamış kanısına varabilirsiniz. Aslında birtakım şeyler oldu. Şimdi elimizin altında neler olduğuna **dir()** fonksiyonu ile bakalım.

```
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__', 'modul7_1']
```

Gördüğünüz gibi modülümüz kullanılabilir durumda. Eğer modül içerisinde, **modül** çağrıldığında işletilecek olan bir satır bulunsaydı bu hemen çalıştırılacaktı. **dir()** iç tanımlı fonksiyonunu yazdığınızda **__<isim>__** şeklinde çeşitli fonksiyonlar karşınıza gelecek. Bunların neler olduğunu şimdilik düşünmeyin, ilerde açıklayacağız.

Şimdi **modul7_1**'in içerisinde neler var neler yok bakalım:

```
>>> dir(modul7_1)
['__builtins__', '__doc__', '__file__', '__name__', 'a', 'isim']
```

Burada gördüğünüz gibi **a** ve isim kullanılabiliyor. Peki bunu nasıl çağıracağız. Eğer bir modülü sadece import ile çağırmışsanız, içerisinde bulunan nesneleri (değişkenler, fonksiyonlar...) şu şekilde çağırabilirsiniz:

<modül_ismi>.<modülde_bulunan_nesne>

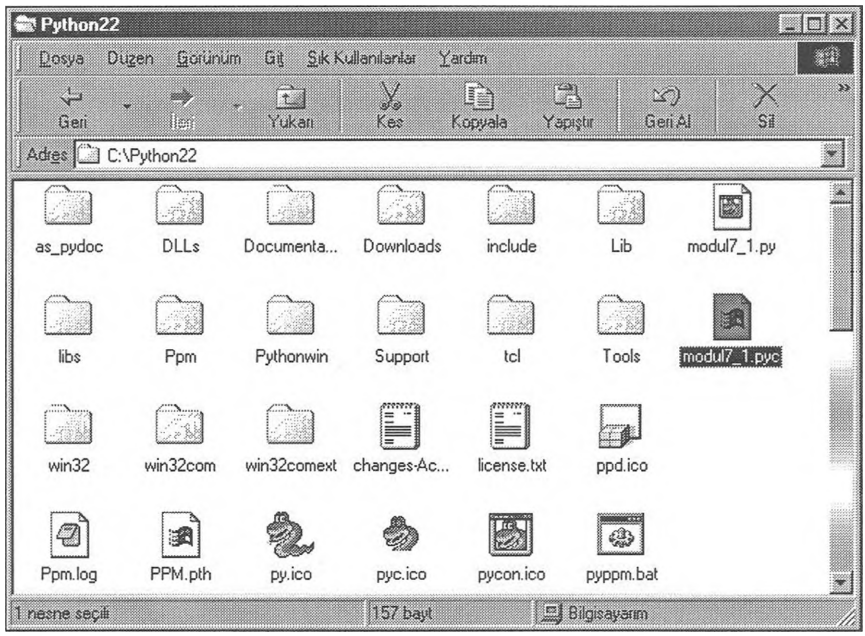
Bu durumda isim nesnesini şöyle çağırabiliriz:

```
>>> ilkmodul.isim
'Mustafa BA\336ER'
```

Gördüğünüz gibi hemen çağrılıp ekrana yazıldı. Bu değişkeni çeşitli amaçlarla kullanabilirsiniz. Örneğin ekrana teker teker yazdırmak için:

```
>>> for harf in ilkmodul.isim: print(harf)
...
M
u
s
t
a
f
a
B
A
Ş
E
R
>>>
```

Şimdi eğer Python'dan çıkıp bulunduğunuz yerdeki dosyalara bakacak olursanız, orada **modul7_1.pyc** isimli bir dosyanın oluştuğunu görürsünüz (**Şekil-7.1**). Bu sizin **modul7_1.py** kaynağınızın byte-code'a derlenmiş halidir. Kaynağı silseniz bile bu modülü kullanabilirsiniz. Bazende **.pyo** dosyalarının var olduğunu görürsünüz. Bunlar da derlenmiş dosyalardır, fakat optimizasyon kullanılmıştır. Optimizasyon yapılırken modül içerisindeki dökümanlar çıkarılmaz. Çoğu zaman optimize edilerek derlenen programların daha verimli olduğu gözlenir.



Şekil-7.1: Modüller çağrıldığında hemen derlenir ve değişiklik olmadığı sürece derlenmiş dosya kullanılır.

Bir modülün optimize edilerek derlenmesini istiyorsanız yorumlayıcı şu şekilde çağırmanız gerekiyor:

```
python -O
```

Artık import ettiğiniz modüller optimize edilerek derlenecektir. Modül optimize edilerek derlenirken aynı zamanda dökümanların da çıkarılması için çift O kullanmalısınız. Modülleriniz bir Python kaynağı (programı tarafından) çağrılırken optimize edilerek derlenmesini istiyorsanız, programınızın ilk satırını şu şekilde değiştirmelisiniz (UN*X kullanıcıları için):

```
#!/usr/bin/python -O
```

Şöyle bir soru aklınıza gelebilir "Ben bu modülü direk çağırmadığıma göre, başta bulunan **#!/usr/bin/python** satırına ihtiyaç var mı?" Hayır, aslında yok. Eğer kod yalnızca ve yalnızca başka programlar tarafından çağrılacaksa ilk satırın bulunmasına gerek yoktur. Ancak çoğu Python programcıları yazdıkları modülün test edilmesi amacı ile, modülü çalıştırılabilir olarak yazar-

lar ve başına bu satırı koyarlar. Ne demek istediğimi şöyle anlatmaya çalışayım: Python modülleri çoğunluka birkaç satırdan oluşmaz, daha karmaşık yapıları ve birçok fonksiyonu içerirler. Bu karmaşık yapının başka bir program tarafından çağırılıp test edilmesi uzun zaman alacağında, genelde her yazılan modül içerisinde, modülü çağırırken gerekli olan verilerden bir örnek bulunur ve modül denenir. Ayrıca, modülü daha sonra kullanmak isteyen kişi de önce bir çalıştırıp ne olup bittiğine bakmak isteyecektir. Bu amaçla siz siz olun yazdığınız modülü test edin ve içerisine örnek bilgiler koyun.

Yukarıdaki modülü biraz değiştirelim ve bir iş yaptıralım **modul7_2.py** olarak kaydedildi):

modul7_2.py

```
#!c:\\python22\\python.exe

a=101
isim="Mustafa BAŞER"
print "Bu modülü çağırırsan bu açıklama yazılır."
print "İlerdeki konularımızda bu açıklamayı yazdırmamayı"
öğreneceğiz."

def denemeFonksiyonu(gelen):
    print "Bu fonksiyona gelen bilgi: ", gelen
```

Python yorumlayıcısını başlatıp modülü çağıralım:

```
>>> import modul7_2
Bu modülü çağırırsan bu açıklama yazılır.
İlerdeki konularımızda bu açıklamayı yazdırmamayı öğreneceğiz.
```

Gördüğünüz gibi modül çağırılır çağrılmaz sanki, bulunduğumuz yerdeki kodlar gibi ilk satırlar çalıştırıldı. Şimdi bu modülde neleri kullanabileceğimize bakalım:

```
>>> dir(modul7_2)
['__builtins__', '__doc__', '__file__', '__name__', 'a',
'denemeFonksiyonu', 'isim']
```

Burada yeni fonksiyonumuzu görüyorsunuz. Hemen bunu çağıralım:

```
>>> modul7_2.denemeFonksiyonu('Muhittin')
Bu fonksiyona gelen bilgi:  Muhittin
```

Bir satırda birden fazla modül çağırmak istiyorsanız isimleri şu şekilde, aralarına virgül koyarak yazabilirsiniz: `import time, os, sys`



Modüllerin Yeniden Çağrılması

Python çağrılan bir modülü ilk seferinde tamamen derler ve kullanılmak üzere bekletir. Modülün **import** komutu ile daha sonra tekrar çağrılması bir anlam ifade etmez, çünkü halihazırda çağrılmıştır. Python kabuğundan çıkmadan modülümüze ikinci bir fonksiyon ekleyelim ve modülü kaydedelim.

Modülün son hali şöyle:

```
#!c:\\python22\\python.exe

a=101
isim="Mustafa BAŞER"
print "Bu modülü çağırırsan bu açıklama yazılır."
print "İlerdeki konularımızda bu açıklamayı yazdırmamayı öğreneceğiz."
def denemeFonksiyonu(gelen):
    print "Bu fonksiyona gelen bilgi: ", gelen

def baskaFonksiyon(gelen):
    print "Burası 'baskaFonksiyon' tarafından yazdırılır."
Gelen veri: ", gelen
```

Şimdi modülü yeniden çağırmaya çalışalım:

```
>>> import modul7_2
```

Gördüğünüz gibi hiçbir tepki yok. Neler olduğuna bakalım:

```
>>> dir(modul7_2)
['__builtins__', '__doc__', '__file__', '__name__', 'a', 'deneme-
Fonksiyonu', 'isim']
```

Bu örneklerde modül değiştirildiği ve tekrar çalıştırıldığı halde, ne ilk satırların yazıldığını ne de eklenen fonksiyonu gördük. Python modülleri sadece bir kez çağırır. Sonraki çağrımlar gözardı edilir. Hem çalışan bir program modülü, etkili olan bir modülü neden tekrar çağırsın ki? Tekrar çağırmak demek programcının bir hatası demektir.

Eğer bir modülü zorunlu olarak tekrar çağırmak istiyorsanız (şu anda bizim yaptığımız gibi modülü sürekli değiştirip, Python'dan çıkmadan yeni değişiklikleri görmek gibi bir nedenle) Python bunu da yapmanıza olanak tanır. Bu amaçla **reload()** fonksiyonu kullanılabilir. İsterseniz hemen deneyelim:

```
>>> reload(modul7_2)
Bu modülü çağırırsan bu açıklama yazılır.
İlerdeki konularımızda bu açıklamayı yazdırmamayı öğreneceğiz.
<module 'modul7_2' from 'c:\windows\desktop\python
kitabı\modul7_2.py'>
>>> dir(modul7_2)
['__builtins__', '__doc__', '__file__', '__name__', 'a',
'baskaFonksiyon', 'denemeFonksiyonu', 'isim']
```

Burada hem ilk satırların çalıştırıldığını hem de yeni fonksiyonun etkili olduğunu görüyorsunuz. Aynı modülü program olarak çalıştıralım: "Edit / Run script"

```
Bu modülü çağırırsan bu açıklama yazılır.
İlerdeki konularımızda bu açıklamayı yazdırmamayı öğreneceğiz.
```

Modüllerin Çağrılış Yöntemleri

Python modülleri, kullanım amacınıza göre değişik şekillerde çağrılabilir. Genel kanı, programın ilk satırında tamamını çağırmaktır. Ancak, modülün birkaç fonksiyonunu kullanacaksanız tamamını çağırıp hafızayı kirletmek pek anlamlı olmayabilir. Aşağıda bir Python modülünü ne zaman ve nasıl çağıracağınızla ilgili ipuçları göreceksiniz.

import <modül>

Bir modülün temel olarak import komutu ile çağrıldığını gördünüz. Bu şekilde çağırdığınız zaman modül içerisindeki nesnelere <modül_ismi>.<nesne> şeklinde ulaşabileceğinizi biliyorsunuz.

Bu şekildeki bir çağrım modülün tamamının hafızaya alınıp kullanılmak üzere bekletilmesidir. Eğer bir modülün sadece birkaç fonksiyonunu kullanacaksanız, size tavsiyemiz bu şekilde çağırmamanızdır.

reload <modül>

Bununla etkin bir modülün zorunluluktan dolayı tekrar çağrıldığını gördünüz. Bu aktif bir modülü tekrar hafızaya alarak modülde yapılan değişikliklerin etkinleşmesini sağlar. Peki bu fonksiyonu nerede kullanabilirsiniz?

Hemen kısa bir cevap verelim. Bazı durumlarda modüller programlar tarafından değişikliğe uğratılabilir ve programın kendi değiştirdiği kodu kendisinin kullanılması kaçınılmaz olabilir. Bu şekilde yazılan dinamik modülleri kullanabilmek için **reload()** fonksiyonu kullanılabilir. Ancak bu tür dinamik modüllerin oluşturulması ileri düzey programcılık bilgisi ister ve çok dikkatli olunması gerekir, zira yapılan değişiklik hemen etkin olacağından koddaki hata programızın çalışmasını durdurabilir. En iyisi böyle bir şeye hiç kalkışmamak.

from <modül> import <nesne>

Modüllerin birçok fonksiyondan (veya sınıftan) oluşabileceğini daha önce söyledik. Özellikle Python ile birlikte dağıtılan modüller birçok fonksiyon içerir. Örneğin bir **re** modülü düzenli ifadelerle ilgili birçok fonksiyondan oluşur ki bunların çoğunu hiç kullanmayız. Bir metinde sadece arama (search) işlemi yapacaksak koskoca bir modülün tamamını çağırmaya gerek yoktur. Sadece **search** nesnesini çağırarak işimizi rahatlıkla görebiliriz. Hem hazıfadan da kazanmış oluruz. Bu gibi durumlarda:

```
from re import search
```

gibi bir çağırımda bulunabiliriz. Şimdi Python yorumlayıcımızı tekrar başlatalım ve **modul7_2.py** modülümüzden sadece **baskaFonksiyon()** fonksiyonunu çağıralım:

```
>>> from modul7_2 import baskaFonksiyon
Bu modülü çağırırsan bu açıklama yazılır.
İlerdeki konularımızda bu açıklamayı yazdırmamayı öğreneceğiz.
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__', 'baskaFonksiyon',
'sys']
```

Bu çağrımın en önemli özelliği ise, modül içerisinden çağırılan nesnenin sanki, o anda bulunduğumuz isim alanında tanımlanış gibi olmasıdır. Bunun anlamı nedir? Eğer **import** komutu ile tüm modülü çağırsaydık, bu modül içerisindeki bir nesneye ulaşmak için **<modül>.<nesne>** şeklinde bir yazılım yapmamız gerekiyordu. Burada ise buna gerek yok, modülden çağırdığımız nesneyi doğrudan kullanabiliyoruz. Yani:

```
>>> baskaFonksiyon('Dilek')
Burası 'baskaFonksiyon' tarafından yazdırılır. Gelen veri: Dilek
```

Bu şekilde çağrımın kod yazılımını kısalttığını da anımsatalım. Ancak bir dezavantajı olduğunu da unutmayın. Bu dezanvantaj nedir? Bulunduğumuz isim alanını kalabalıklaştırmış oluyoruz (İsim alanı ile ilgili geniş açıklamayı sınıflar bölümünde bulacaksınız). Eğer bir modülden bu on şekilde tane fonksiyon çağırırsanız ortalık arap saçına döner ve de çağırdığınız nesne ismi ile yeni bir nesne tanımlama imkanınız olmaz. Veya aynı isimde başka modüllerin nesneleri olabilir. Örneğin **`string`** modülünden **replace** fonksiyonunu bu şekilde çağırırsanız **re** modülünde bulunan **replace** fonksiyonunu çağırdığınızda önceki fonksiyon yitirilmiş olur. Bunu önlemek için bir modüldeki bir nesneyi başka bir isim ile çağırabileceğiniz "as" yöntemini kullanabilirsiniz (aşağıda). Tüm modülü çağırdığınızda, **`replace`** nesnesi **`string`** modülüne ait olduğundan bulunduğunuz yerde **replace** ismi ile yeni bir nesne tanımlayabilirsiniz. Genelde tavsiye ettiğimiz bir yöntemdir.

from <modül> import *

Aslında bu çağrımın yukarıdakinden tek farkı, modül içerisindeki bulunan tüm nesnelerin yerel isim alanına çağrılmasıdır. Örneğin time modülünü

```
>>> form time import *
```

şeklinde çağırırsanız, bu modül içerisindeki tüm fonksiyonları sanki o anda bulunduğunuz isim alanında tanımlanmış gibi kullanabilirsiniz:

```
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__']
>>> from time import *
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__', 'accept2dyear',
'altzone', 'asctime', 'clock', 'ctime', 'daylight', 'gmtime', 'lo-
```

```
caltime', 'mktime', 'sleep', 'strftime', 'time', 'timezone',
'tzname']
>>> print ctime(time())
Mon Dec 31 23:18:21 2001
```

from <modül> import <nesne> as <yeni_isim>

Bu çağırım ile herhangi bir modüldeki herhangi bir nesneyi yeni vereceğiniz isim ile çağırabilirsiniz. Örneğin:

```
>>> from time import ctime as CZamani
>>> CZamani(1009833478.69)
'Mon Dec 31 23:17:58 2001'
```

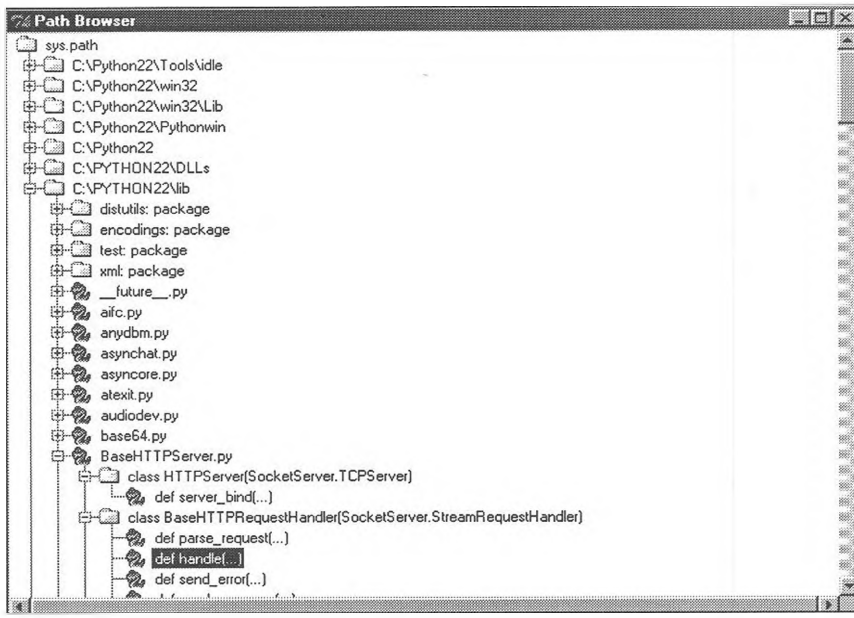
import <modül> as <yeni_isim>

Bu tip çağrımlarda içereceğiniz modülü kendi vereceğiniz bir isim ile çağırmış olursunuz:

```
>>> import time as zaman
>>> zaman.ctime(zaman.time())
'Mon Dec 31 23:24:22 2001'
```

Hangi Modülde Hangi Nesneler Var?

Modüller içerisindeki nesneleri bilmek için mutlaka import edip sonra da **dir()** çekmenize gerek yok. Idle yazarı (aynı zamanda Python'un da yazarı) modüller içerisindeki nesneleri bilmenin çok önemli olduğunun bilincindedir ve bu yüzden size harika bir tarayıcı sunuyor. Tarayıcıya ulaşmak için "File / Path browser" yolunu izleyin. **Şekil-7.2**'de örnek bir pencere görüyorsunuz:



Sekil-7.2: Python modüllerinde bulunan nesneleri Path Browser yardımı ile kolayca bulabilirsiniz.

Özel İsim ve Fonksiyonlar: __isim__

Python'da sınıflara ait birçok özel isim ve fonksiyon vardır. Bunlar ikişer tane alttan çizgi arasına yazılır.

Program İsimleri : __name__

Her programın bir ismi vardır. Bu isim modül ismi ya da çalışan programın dosya ismi değildir. Çalışan her programın kendisine ait olan isim **__main__**'dir. Örneğin hiçbir modül çağrılmadan Python yorumlayıcısında **dir()** yazarsanız **__name__** nesnesini göreceksiniz.

```
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__']
```

Peki bu sonucun anlamı nedir? İsterseniz **dir()** fonksiyonunun nasıl bir veri tipine sahip olduğuna bakalım:

```
>>> type(dir())
<type 'list'>
```

Demek ki bir liste veriyor bize. O halde bu liste elemanlarının bir de değeri vardır. Şimdi **__name__** elemanının değerine bakalım:

```
>>> __name__
'__main__'
```

Yani bir program çalıştırıyorum ve bunun ismi **__main__**. Python komut satırı da bir program gibidir ve ismi **__main__**'dir. Şimdi **modul7_2**'yi çağıralım ve onun ismine bakalım:

```
>>> dir(modul7_2)
['__builtins__', '__doc__', '__file__', '__name__', 'a',
'baskaFonksiyon', 'denemeFonksiyonu', 'isim']
```

Buradaki **modul7_2** çıktısı bu modülün ismini veriyor (dosya ismini değil). Bu isme ait nesneler vardır ve o nesneler bu isim altından çağrılabilir. Şimdi bir **modul7_2.py** dosyasını değiştirelim ve bunun modül olarak çağrıldığında bazı işleri yapmasını önleyelim. Ancak kendi başına çalıştırıldığında yine belirlenen işleri yapsın. Yukarıda gördüğümüz gibi bir programın kendi içerisindeki ismi **__main__**'dir. O halde bir **if** ifadesi ile, program kendisi mi çalıştırılıyor yoksa modül olarak mı çağrılıyor ayırt edebiliriz. Bu açıklamalar altında modülümüzü şöyle değiştirelim (dosyayı **modul7_3.py** olarak kaydettik):

modul7_3.py

```
#!c:\\python22\\python.exe

a=101
isim="Mustafa BAŞER"

def denemeFonksiyonu(gelen):
    print "Bu fonksiyona gelen bilgi: ", gelen
def baskaFonksiyon(gelen):
    print "Burası 'baskaFonksiyon' tarafından yazdırılır. Gelen
veri: ", gelen

if __name__=='__main__':
    print "Bu programın kendisini çalıştırırsan bu açıklama
yazılır."
    print "Modül olarak çağırırsan, hiçbir açıklama yazılmaz."
```

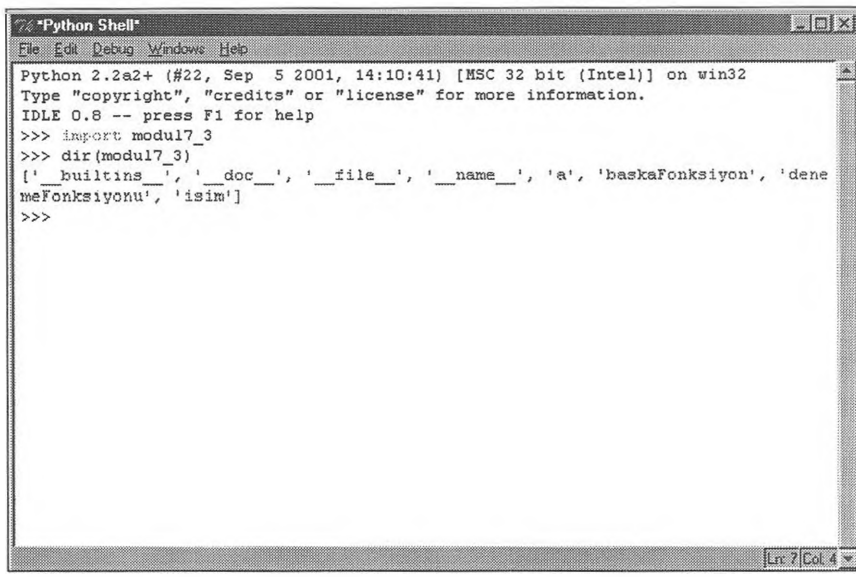
Şimdi Python yorumlayıcısını çalıştırıp bu modülü çağıralım:

```
>>> import modul7_3
```

Gördüğünüz gibi herhangi bir satır çalıştırılmadı. İsim alanımıza bakalım:

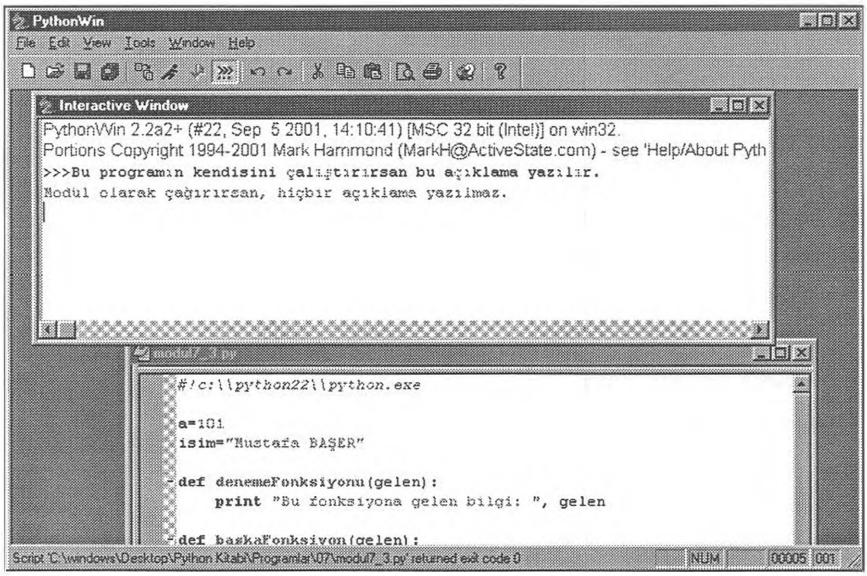
```
>>> dir()
['PyShell', '__builtins__', '__doc__', '__name__', 'modul7_3',
'sys']
>>> dir(modul7_3)
['__builtins__', '__doc__', '__file__', '__name__', 'a', 'baska-
Fonksiyon', 'denemeFonksiyonu', 'isim']
```

Tüm bu yaptıklarımızın sonucunu **Şekil-7.3**'de görüyorsunuz.



Şekil-7.3: Modül ve program ayrımını yapabilirsiniz.

Buradan da anlaşılacağı üzere modülümüz çağrılmış ve nesneler oracıkta duruyor. Bu modülü program olarak çalıştırırdığımızdaki sonucu **Şekil-7.4**'te görüyorsunuz.



Şekil-7.4: Python bir programı çalıştırdığında modül olarak mı yoksa kendisinin mi çağrıldığını anlayabilir.

Dosya İsimleri: __file__

Yukarıda gördüğünüz üzere bir de **__file__** elemanı var. Bu özel nesne, modülün dosya ismini saklar.

```
>>> modul7_3.__file__
'C:\\Python22\\modul7_3.pyc'
```

Burada gördüğünüz gibi derlenmiş olan modül çağrılıyor.

Açıklama Satırları: __doc__

Python o kadar ince ve ayrıntılı düşünülmüş ki, her modül kendi içerisinde açıklamaları (dokümanları) içerir. Hatta modül içerisindeki herhangi bir nesne altında da açıklamalar mevcut olabilir. Fonksiyonlar bölümünde kısa bir açıklama yaptığımız gibi, başınız sıkıştığında hemen bu dokümanlara başvurabilirsiniz, elbette modül yazarı koymuşsa. Açıklama satırları üç tırnak arasına ve ilk satırlar olarak yazılır. Şimdi kendi yazdığımız modül içerisine açıklamalar koyalım. Aşağıdaki gibi bir modül yazıp, **modul7_4.py** olarak kaydedin.

modul7_4.py

```
#!c:\\python22\\python.exe

"""
Bu modül eğitim amaçlı yazılmıştır.
Herhangi özel bir işlevi yoktur.
"""

from math import sqrt
from sys import argv


def ikinciDereceDenklemleri(a,b,c):
    """

    Bu fonksiyon, ax^2 + bx + c = 0 şeklinde verilen
    ikinci derece denklemini köklerini verir. Fonksiyona
    sağlanacak olan parametreler sırası ile a, b, c olmalıdır.

   """

   if a==0:
      print "İkinci derece denklemin a katsayısı sıfır olamaz."
   else:
      delta=b*b-4*a*c
   if delta==0:
      print "Bu denklemin eşlenik iki kökü vardır"
      print "X1,X2 : ", b/(2.0*a)
      print
   elif delta < 0:
      print "Bu denklemin karmaşık iki kökü vardır"
      delta=-delta
      deltas=sqrt(delta)
      print "X1:", b/2.0*a, " - i", delta/2.0*a
      print "X1:", b/2.0*a, " + i", delta/2.0*a
```

```
        print
    else:
        print "Bu denklemin iki gerçek kökü vardır"
        deltas=sqrt(delta)
        print "X1:", (b - delta)/2.0*a
        print "X1:", (b + delta)/2.0*a
        print

if __name__=="__main__":

    if len(argv) > 3:
        ikinciDereceDenklemleri(int(argv[1]),int(argv[2]),
int(argv[3]))
    elif len(argv)==2 and argv[1]=='--test':
        print "Program test için çağrılmıştır. Şimdi üç deneme yapalım:"
        ikinciDereceDenklemleri(1,-4,4)
        ikinciDereceDenklemleri(2,16,3)
        ikinciDereceDenklemleri(1,2,3)
    else:
        print "Kullanılışı:"
        print "Test etmek için: --test"
        print """İkinci derece denklemin köklerini bulmak için
katsayılarını a b c şeklinde argüman olarak verin"""
```

Yukarıdaki Python kodu bir modülde bulunması gereken temel nesneleri içeriyor. Şimdi bu programı çalıştırırsak aynen aşağıdaki şekildeki gibi olur.

Örnek Modülün Çalıştırılması

Bu modülü Python yorumlayıcısı içerisinden çağıralım. Bu modülün başka bir program tarafından çalıştırılması gibidir. Buna da bir örnek vereceğiz.

```
>>> import modul7_4
>>> print modul7_4.__doc__

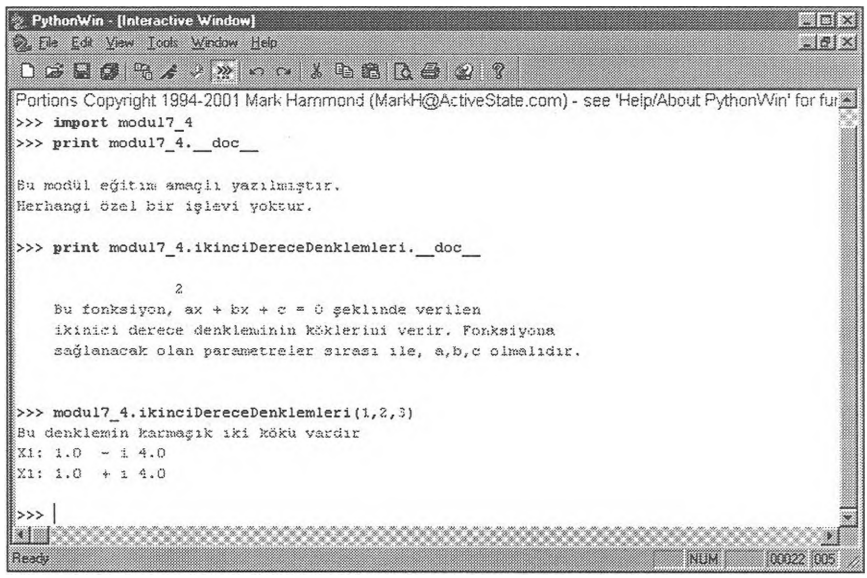
Bu modül eğitim amaçlı yazılmıştır.
Herhangi özel bir işlevi yoktur.
```

```
>>> dir(modul7_4)
['__builtins__', '__doc__', '__file__', '__name__', 'argv',
'ikinciDereceDenklemleri', 'sqrt']
>>> print modul7_4.ikinciDereceDenklemleri.__doc__

    Bu fonksiyon, ax² + bx² + c = 0 şeklinde verilen
    ikinci derece denklemin köklerini verir. Fonksiyona
    sağlanacak olan parametreler sırası ile a, b, c olmalıdır.

>>> modul7_4.ikinciDereceDenklemleri(1,2,3)
Bu denklemin karmaşık iki kökü vardır
X1: 1.0  - i 4.0
X1: 1.0  + i 4.0
```

Yaptıklarımızı **Şekil-7.5**'de görüyorsunuz.



Şekil-7.5: Gerçek anlamda ilk modülümüzün çağırılıp kullanılması

Modüllerim Nerede?

Python modülleri çağırırken **path** çevre değişkeninizde bulunan dizinlerde arayıp çağırmaya çalışacaktır. Bulunduğunuz klasör her zaman çevre değişkeni içerisine alınır. Dolayısıyla bulunduğunuz yerden çağırdığınızda herhangi bir işlem yapmanıza gerek kalmaz. Ancak modüller her zaman bulun-

duğunuz yerde veya **path** çevre değişkenininde tanımlı klasörlerde olmayabilir. Sizin **path**'iniz ile başka bir kullanıcınınki farklı olabilir (ya da CGI için, web sunucusununki). O zaman çağırılacak modüllerin bulunduğu dizini **path**'e eklemek gerekir. Modül çağrılmadan önce **sys** modülü çağırılmalıdır. Örneğin:

```
>>> import sys
>>> sys.path.append('c:/windows/desktop/python kitabı')
>>> mport modul7_4
```

Bir Modülü Ne Zaman Çağırmalı?

Hemen aklınıza bu soru gelecektir. Genel eğilim modüllerin programın ilk başında çağırılmasıdır, ki çağırılan modül tüm isim alanında etkili olsun. Ancak bazı modüller program akışı sırasında her zaman kullanılmayabilir. Sadece bir koşul sağlandığında gerekebilir. Bu durumda modülü o anda çağırmak daha iyi olacaktır. Böylelikle hem hafızadan tasarruf edilmiş olacak hem de program açılışı hızlanacaktır.

Tüm Modüllerimi Derlemek İstiyorum

Yazdığınız program birçok modülden oluşuyorsa, modülleri çağırmadan önce hepsini derleme ihtiyacı hissedebilirsiniz. Bunun için **compileall** modülü kullanılabilir. Bunu şu şekilde kullanablirsiniz.

```
>>> compileall.compile_dir('c:\\windows\\desktop\python kitabı')
Listing c:\windows\desktop\python kitabı ...
Compiling c:\windows\desktop\python kitabı\modul7_1.py ...
Compiling c:\windows\desktop\python kitabı\modul7_2.py ...
Compiling c:\windows\desktop\python kitabı\modul7_3.py ...
Compiling c:\windows\desktop\python kitabı\modul7_4.py ...
Compiling c:\windows\desktop\python kitabı\my.py ...
Compiling c:\windows\desktop\python kitabı\program5-1.py ...
```

Eğer tüm modüller doğru bir şekilde derlenebilirse sonuç 1, derlenemezse sonuç 0 olur. Unutmayın, derlenirken optimizasyon yapılmasını istiyorsanız Python yorumlayıcısını "-O" parametresi ile çalıştırın.

Hatalarla Başetme

08

Şimdiye kadar bazı örneklerimizde hata uyarılarını gördük; fakat bunları gözardı ettik. İyi bir programcı yazdığı kodun mümkün olan her şartta çalışmasını ister. Olası hatalara karşı hazırlıklı olur ve bunu kod yazılımına yansıtır. Hatalar her zaman kullanıcının yaptığı yanlışlıklardan doğmaz, bazen gözardı edilen bir yerde hata olma olasılığı vardır. Ancak çoğu zaman hataların doğabileceği yerler iyi programcılar tarafından fark edilir ve önlemleri alınır. Aşağıda kod yazılışından kaynaklanan bir hatayı görüyorsunuz:

```
>>> for k in range(9)
  File "<stdin>", line 1
    for k in range(9)
                   ^
SyntaxError: invalid syntax
>>>
```

Biz bu bölümde kod yazılışından doğacak hatalardan bahsetmeyeceğiz. Nasıl olsa her programcı yazdığı kodu inceler ve programını olası değerler için çalıştırarak bu tür hatalardan kendisini koruyabilir. Ancak hataların çoğu bu tür yanlış yazılıştan kaynaklanmaz. Aksine, bir değişkenin alabileceği değerin tasarlanandan farklı olması durumunda ortaya çıkacak hatalar gibisi daha çoktur. Hemen buna bir örnek verelim:

program8_1.py

```
#!c:\\python22\\python.exe

a=raw_input('Bir Sayı giriniz: ')

print "Girdiğiniz sayının karesi:", int(a)*int(a)
```

Bu programı çalıştırdığımızda kullanıcının ne gireceğinden emin olamayız. Hemen yanlış kullanımı görelim:

```
Bir Sayı giriniz: p
Girdiğiniz sayının karesi:
Traceback (most recent call last):
  File "C:/windows/Desktop/Python Kitabı/program8-1.py", line 5,
in ?
    print "Girdiğiniz sayının karesi:",  int(a)*int(a)
ValueError: invalid literal for int(): p
```

Gördüğünüz gibi, karesinin alınması amacı ile kullanıcıdan bir sayı istiyoruz, o ise sayı yerine bir harf giriyor. Eğer programınız uzunsa ve sırf bu yüzden hata verip akış durduruluyorsa, bu kullanıcının değil programı yazanın hatasıdır. Kullanıcının vereceği bilgilerin ne olacağından emin olamazsınız. O halde bu gibi durumlarda önlem almak kaçınılmazdır. Böyle hataların ortaya çıkışına Python'da "hata yükselmesi" denir.

Bu bölümde yukarıdaki gibi hataları nasıl yakalayacağımızı ve bunlarla nasıl başa çıkacağımızı göreceğiz. Programınızı modüler olarak yazmışsanız, yazdığınız modüller başkaları tarafından da kullanılacaksa, bu gibi bir hata ile karşılaştığınızda sizin de bir hatayı ön plana çıkarma gereksiniminiz olacaktır. Bölümün en sonunda bunu nasıl yapacağımızı göreceğiz.

try-except

İyi yazılmış Python programlarında çokça karşılaşılan bu deyim, "Bir hata ile karşılaşana kadar dene" anlamına geliyor. Örneğin bir cümleyi sayı değere dönüştürmeye çalışırken hatalardan korunmak için şöyle bir yol izleyebilirsiniz:

```
>>> a="python"
>>> try:
...     int(a)
... except:
...     print "Uyarı: 'a' değişkeninin şu andaki değeri sayıya \
çevrilemiyor!"

Uyarı: a değişkeninin şu andaki değeri sayıya çevrilemiyor!
```

Bu örnekte **a** değişkenine önce Python değerini atadık. Bu değeri **int()** hazır fonksiyonu ile sayıya çevirmeye çalışıyoruz. Ancak cümle 0-9 arasındaki karakterlerden oluşuyorsa sayıya çevrilebilir. **a** değişkeni sayıya çevrilemediğinden bir hata yükseliyor. Yükselen hata **except** ile yakalanıp hemen altındaki blok işletiliyor. Bu blokta ise değerin sayıya çevrilemediği kullanıcıya bildiriliyor.

Ön tanımlı yükselebilecek hataların tam listesini *Library Reference*'den görebilirsiniz.

Öğrendiğimiz bilgiler ışığında yukarıdaki örneği tekrar yazalım:

program8_2.py

```
#!c:\\python22\\python.exe

a=raw_input('Bir Sayı giriniz: ')

try:
    print "Girdiğiniz sayının karesi:",  int(a)*int(a)
except:
    print "Girdiğiniz değer sayıya çevirilemiyor."
```

Eğer bir hata ile karşılaşıldığında herhangi bir işlem yapılmadan sonraki satırın işletilmesine devam etmek istiyorsanız **pass** ifadesini kullanabilirsiniz. Kullanımına şöyle bir örnek verelim:

```
>>> Sayilar=["5","9","f","2","t","6"]
>>> for i in Sayilar:
...     try:
...             sayi=int(i)
...             print i, "sayısının karesi: ", sayi*sayi
...     except:
...             pass
...
5 sayısının karesi:  25
```

```
9 sayısının karesi:  81
2 sayısının karesi:  4
6 sayısının karesi:  36
```

Burada görüldüğü gibi sayilar listesinde sayı dışı bazı elemanlar bulunmakta fakat yazdığımız döngü hatalarla başa çıktı ve liste elemanlarının tümü üzerinde işlemler yapıldı.

Hata Uyarılarını Belirleme

Bir hata yükseldiğinde çoğu zaman o hatanın ne olduğu da belirtilir. Kullanıcıya yanlışının ne olduğunu söyleyebilmek için yükselen hatadan haberdar olmak gerekir. **except** ifadesini kullanırken hangi hatalarla başa çıkacağınızı da belirleyebilirsiniz. Yazılışı şu şekildedir:

try:
yapılacak işler
excpet hata_kodu 1:
hta_kodu 1 yükseldiğindeki davranış
except hata_kodu 2:
hata_kodu 2 yükseldiğindeki davranış
except:
diğer hatalardaki davranış

Bu yazılımdan da anlaşılacağı üzere, yazdığınız program yükselen hatanın cinsine göre davranışını değiştirmektedir. Şimdi bir değişkenin sayıya çevrilmesine tekrar bakalım:

```
>>> b="k"
>>> int(b)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in ?
    int(b)
ValueError: invalid literal for int(): k
>>> b=(1,2)
>>> int(b)
```

```
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in ?
    int(b)
TypeError: object can't be converted to int
```

Buradaki örnekte birinci hata ile zaten tanışığız; fakat ikinci hatayı daha önce hiç görmedik. İlkinde bir harf sayıya çevrilmeye çalışılıyor ve "**Value Error**" yükseliyor. İkincisinde ise bir tüp değişken sayıya çevrilmek isteniyor ve "Type Error" yükseliyor. Demek ki aynı fonksiyon birden fazla çeşit hata verebiliyor. Bu durumda kullanıcıya tam olarak hatanın kaynağını söylemek ya da program akışını hatanın cinsine göre değiştirmek gerekiyor. Şimdi böyle bir önlem alalım:

```
>>> a=["2","b",5,(9,7),"11"]
>>> for i in a:
...     try:
...         sayi=int(i)
...         print i, "sayısının karesi: ", sayi*sayi
...     except ValueError:
...         print "Sayı değişkene çevrilemeyen karakter: ", i
...         pass
...     except TypeError:
...         print "Sayı değişkene çevrilemeyen değişken tipi: ",i
...
2 sayısının karesi:  4
Sayı değişkene çevrilemeyen karakter:  b
5 sayısının karesi:  25
Sayı değişkene çevrilemeyen değişken tipi:  (9, 7)
11 sayısının karesi:  121
```

Hataları Kullanıcıya Bildirme

Programı kullanan kişi karşılaştığı hatanın ne olduğunu tam olarak anlarsa, onun önlemini almaya çalışır. Dosyaları okuyup çeşitli işlerin yapılacağı bir program kodu içerisinde dosyanın açıldığı yere bakalım:

*(UN*X ve NT için)*

```
>>> dosya = open('/deneme2.txt')
```

```
Traceback (innermost last):
  File "<stdin>", line 1, in ?
IOError: [Errno 13] Permission denied: ' /deneme2.txt'
```

Burada okuma hakkının bulunmadığı bir dosya okunmak üzere açılmaya çalışılmıştır. Şimdi de olmayan bir dosyayı okumak üzere açmaya çalışalım:

```
>>> dosya = open('c:\\deneme.txt')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in ?
    dosya = open('c:\\deneme.txt')
IOError: [Errno 2] No such file or directory: 'c:\\deneme.txt'
```

Eğer şöyle bir kod yazarsak, kullanıcı dosyanın niye açılamadığını anlayamaz:

program8_3.py

```
#!c:\\python22\\python.exe

print "Okunmak üzere dosya ismi giriniz: "

dosyaIsmi=raw_input()

try:
    dosya=open(dosyaIsmi)
except:
    print dosyaIsmi, "dosyası açılamıyor."
```

Bu programı çalıştırırsak:

```
>>>
Okunmak üzere dosya ismi giriniz:
c:/deneme.txt
c:/deneme.txt  dosyası açılamıyor.
```

Gördüğünüz gibi bir hata ile karşlaşılıyor ancak hatanın ne olduğu belli değil. Yükselen hataları kullanıcıya açık seçik belirtmek gerekiyor. Elbetteki

sadece kullanıcıya bildirmeniz değil, program akışı içerisinde hataların tam olarak neden yükseldiğini anlayıp ona göre önleminizi almanız, akışı değiştirmeniz gerekiyor.

Yükselen hataların değerlerini şöyle elde edebiliriz:

***except** hata_kodu, (hata_no, hata_anlami):*

2 numaralı hatanının dosyanın olmadığı durumda, 13 numaralı hatanın ise okuma hakkımızın olmadığı durumda ortaya çıktığını biliyoruz. Şimdi bu öğrendiklerimizi programımız da uygulayalım:

program8_4.py

```
#!c:\\python22\\python.exe
print "Okunmak üzere dosya ismi giriniz: "

dosyaIsmi=raw_input()

try:
    dosya=open(dosyaIsmi)
except IOError, (hata_no, hata):
    if hata_no==2: print dosyaIsmi, "isimli dosya bulunmuyor"
    if hata_no==13: print dosyaIsmi, "isimli dosyayı okuma \
hakkınız yok"
```

Burada iki hatada aynı cinsten (IOError) olup sadece hata kodları farklıdır. Bazen öyle olur ki, farklı hata mesajları yükselebilir. Örneğin eğer var olmayan bir dosyayı okunmak üzere açmaya çalışırsanız "IOError" hatası yükselir. Dosya var, fakat okuduğunuz satırı sayı değişkene çevirmeye çalışıyorsunuz, bu durumda da "ValueError" yükselecektir. Bunların hepsine hazırlıklı olan bir programcının yazacağı kod şuna benzeyecektir:

program8_5.py

```
#!c:\\python22\\python.exe
import string
```

```
satir=1
while satir:
    try:
        dosya=open('c:\\tmp\\deneme.txt')
        satir=dosya.readline()
        karesi=int(satir)*int(satir)
    except IOError:
        print "deneme.txt dosyası açılamıyor"
        satir=0
    except ValueError:
        print "Okunan satır sayıya çevrilemiyor"
    except:
        print "Beklenmeyen hata"
```

Dikkat ederseniz yukarıdaki programda beklenen iki hata için kullanıcı uyarılıyor. 10. satırda eğer dosya açılamazsa (kullanıcının okuma izni olmayabilir, dosya yerinde değildir...) bu durum kendisine bildiriliyor. 13. satırda ise, sayı değere çevrilemeyen bir satır okunursa mesaj bildirecektir. Bunların dışındaki hatalar ise beklenmedik hatalar olarak nitelendiriliyor.

Soru: Yukarıdaki programda 14. satırda satir=0 ifadesi neden yazılmıştır?

Cevap: Eğer dosya okunmak üzere açılamazsa ve satir=0 ifadesi bulunmazsa bu durumda 6. satırdaki karşılaştırma her zaman doğru olacağından program sonsuz bir döngüye girmiş olur. satir=0 demekle, karşılaştırmayı yanlış yapmış oluyor ve döngüyü sonlandırıyoruz.

Soru: Beklenmeyen hata ne olabilir?

Cevap: 10 ve 13. satırlardaki hatalar dışında yükselmeyen başka hatalar olabilir. Örneğin herhangi bir satırda bulunan değer sayıya çevrilip kendisi ile çarpıldığında o kadar büyük olur ki "OverflowError" hatası yükselebilir.

try-except-else

Python programlama dilinin harika dünyasındaki gezimize devam ediyoruz. Bazen hata yakalamak istediğimiz zaman, bir hata yükselmezse belirli bir fonksiyonu ya da bloğu çalıştırmak durumunda kalabiliriz. Bu durumda **try-except-else** ifadesini kullanmalısınız.

Örneğin bir CGI programı düşünelim. İyi yazılmış bir web programı, kullanıcıdan gelen verinin boyutunu kontrol edererek sunucu güvenliğini sağlama alır. Bu durumda kullanıcıdan gelen verinin boyutu belirleneni aşarsa, hata verilmeli yoksa işleme devam edilmelidir. İşte aşağıdaki programda bunu görüyorsunuz:

program8_6.py

```
#!c:\\python22\\python.exe

print "Content-type: text/html"
print

try:
     import cgi
     cgi.maxlen=20*1024
     form=cgi.FieldStorage()
except:
     print """
     <h1>
     Sunucya fazla bilgi gönderiyorsunuz,işleminiz iptal edidi.
     </h1>
     """

else:
     print "Şimdi işleminize devam edilecek"
     # Program kodları bu bloğa yazılacak
```

Yukarıdaki programda **try** bloğunda ilk olarak **cgi** modülü içeriliyor. Sonra ilk yapılması gereken şey gelen verinin boyutunu irdeleyen, **cgi.maxlen** ile alınacak en büyük verinin 20K olması sağlanıyor. Eğer sınır aşılmamışsa, **form=cgi.FieldStorage()** satırı ile kullanıcı verileri form nesnesine, daha sonra kolayca çağrılacak bir biçimde alınıyor. **cgi.maxlen** satırında bir hata yükselirse hemen **except** bloğu devreye giriyor ve işlem kesiliyor. Herşey yolundaysa **else** bloğu ile program akışına devam ediliyor.

Hata Yükseltme

İyi yazılmış Python modülleri bir olumsuzlukla karşılaştığında mantıklı hatalar yükseltirler. Yükselen hatalar standart Python hataları olabildiği gibi

programcı tarafından tanımlanmış da olabilir. Aşağıdaki **H()** foksiyonu çağırıldığında böyle bir hata yükseliyor:

program8_7.py

```
#!c:/python22/python.exe

hataDegiskeni=''
h=''

def H():
    raise hataDegiskeni, "Bu hata çok tehlikeli"

try:
       H()
except hataDegiskeni, h:
       print "Bizim hata Yükseldi:", h

except:
       print "Başka bir hata yükseldi"
```

Program çalıştırıldığında şu sonucu veriyor:

```
Bizim hata Yükseldi: Bu hata çok tehlikeli
```

Yukarıdaki programı şu şekilde de yazabilirsiniz:

program8_8.py

```
#!c:/python22/python.exe

h=''

def H():
    raise "benimHatam", "Bu hata çok tehlikeli"

try:
```

```
        H()
except "benimHatam", h:
        print "Bizim hata Yükseldi:", h

except:
        print "Başka bir hata yükseldi"
```

Düzenli İfadeler

09

Yazdığınız bir programda mutlaka metin içerisinde istediğiniz bir deseni arama işlemi yaparsınız. Özellikle büyük metinleri işliyorsanız, desenleriniz çok çeşitli oluyorsa, her desen çeşidi için birer fonksiyon yazmak oldukça baş ağrıtıcıdır. Eski bir Pascal programcısı olarak bunun ne kadar zahmetli bir iş olduğunu çok iyi bilirim. Modern bir programlama dili kullanmıyorsanız şu anda aynı dertleri yaşıyor olmalısınız. Örneğin içerisinde 'ş' harfi bulunan fakat 'ğ' bulunmayan kelimeleri arıyorsunuz. Bunun için bir fonksiyon yazmalısınız. Ya da 'Fr' ile başlayıp, '>.' ile biten bir satır bulmaya ihtiyacınız var. Bunun için de yine bir fonksiyon yazmak durumundasınız. İçerisinde Türkçe harflerin bulunduğu bir veriyi kabul etmek istemiyorsunuz. Bu durumda yine ayrı bir fonkisyon yazacaksınız. Eğer Perl biliyorsanız bunların ne kadar da kolay yapıldığını söylercesine gülümsüyor olmalısınız. Python aynen Perl gibi güçlü bir "düzenli ifadeler" (regular expressions) modülüne sahiptir. Bu bölümde programcılık hayatınızı oldukça kolaylaştıracak bilgileri öğreneceksiniz.

Python'un 1.5'ten sonraki sürümleri Perl tipi düzenli ifadeler modülü olan **re**'ye sahiptir. Önceki sürümlerinde ise Emacs yazılımlı **regex** modülü kullanılırdı. **regex** hem okunabilir değildir hem de pek fazla özellik sunmaz. Program yazarken kesinlikle **re** modülünü kullanmamalısınız. Ancak eski kodları inceleyecekseniz bir şekilde **regex** modülünü de öğrenme ihtiyacı hissedeceksiniz. Kitabımızda **regex**'ten bahsetmeyeceğiz.

Python'da düzenli ifadeler önce bir dizi byte-code'da derlenir, daha sonra C ile yazılmış bir motor tarafından irdelenir.

Re oldukça karmaşıktır. Ünlü bir bilgisayar yazarı olan Jamie Zawinski şöyle diyor:

"Bazı insanlar bir problemle karşılaştıklarında şöyle düşünür: 'Biliyorum, şimdi düzenli ifadeleri kullanmam gerekiyor'. Evet, şimdi onların iki tane problemi var."

Buna rağmen düzenli ifadelerden kurtuluş yoktur. Biz en basittten başlayarak zora doğru gideceğiz.

Karakter Eşleme

Konumuza en basit iş olan karakter eşleme ile başlıyoruz. Bir cümle ya da metin içerisinde bir karakterin var olup olmadığını belirlemek kuşkusuz çok önemlidir. Bir cümle ya da metnin bir dizi karkterden herhangi birisine sahip olup olmadığını da belirleyebilirsiniz. Python eşleme motoru ön tanımlı olarak büyük-küçük harfe karşı duyarlıdır. Yani 'benim' ile 'Benim' birbirlerine eşlenmezler. Ancak bu duyarlılığı etkisiz hale getirip eşlemeyi doğru kılabilirsiniz.

re modülünde çeşitli fonksiyonlar vardır. Bunlardan en çok kullanılanı **match()**, diğeri de **search()**'tür. Eğer aradığınız desen metnin ilk başında ise **match()** fonksiyonun, aralarda ise **search()** fonksiyonunu kullanacağız. Her iki fonksiyon da aranan desen bulunduğunda nesnenin hafızadaki yerini döndürecek, bulunmazsa hiçbir şey döndürmeyecektir. Şimdi hemen bir örnek verelim:

```
>>> import re
>>> re.match("F","Fatih")
<SRE_Match object at 0x01417050>
>>> re.match("F","Melike")
>>>
```

Her iki eşlemede de cümlenin ilk harfinin "F" olup olmadığına bakıyoruz. Birincisinde sonuç doğru ve nesnenin yeri veriliyor, ikincisindeyse yanlış ve birşey verilmiyor. Aslında bu kadar basit bir iş yapacaksanız **re** modülünü kullanmanıza gerek kalmadan

```
>>> "Fatih"[0]=="F"
```

gibi bir ifade de kullanabilirdiniz. Şimdi ilk harfin "F" veya "M" olduğunu denetleyelim. Bu tür denetimler için desenleri [] arasına koymamız gerekiyor.

```
>>> re.match("[FM]","Fatih")
<SRE_Match object at 0x00AA7240>
>>> re.match("[FM]","Melike")
```

```
<SRE_Match object at 0x00AE0FC0>
>>> re.match("[FM]","Dilek")
>>>
```

Şimdi yavaş yavaş düzenli ifadeleri etkin bir şekilde kullanmaya başladık. Gördüğünü gibi ilk iki eşleme doğruyken sonraki yanlış. [] arasına bir dizi de yazabilirsiniz. Örneğin ilk karakterin bir rakam olup olmadığını sınamak için "[0-9]" kullanabilirsiniz. Buradaki tire (-) işaretinin anlamı 0'dan 9'a kadardır. Bunu örneklersek:

```
>>> re.match("[0-9]","Mustafa")
>>> re.match("[0-9]","9.uncu Cumhurbaşkanı")
<SRE_Match object at 0x01431FC0>
```

"[a-z]" küçük harfleri ifade eder:

```
>>> re.match("[a-z]","Dilek")
>>> re.match("[a-z]","dilek")
<SRE_Match object at 0x0143AD60>
```

Benzer şekilde "[A-Z]" büyük harflerdir. Tüm bunları birleştirerek de kullanabilirsiniz. Yani "[0-9a-zA-Z]" gibi. Şimdi tüm bunlarla **search()** fonksiyonunu kullanalım. Çoğu zaman bir cümlenin sadece harf ve rakamlardan oluşup oluşmadığını kontrol ederiz. Bu durumda bir olumsuzluktan söz ediyoruz; **re**'de olumsuzluklar "^" ile belirtilir. Daha iyi anlamak için şu örneğe bakalım:

```
>>> re.search("[^0-9a-zA-Z]", "MustafaBaser")
>>> re.search("[^0-9a-zA-Z]", "Mustafa Baser")
<SRE_Match object at 0x01455570>
```

ikincisinde bir boşluk olduğu için eşleme sonucu verildi. Şimdi şuna bakalım:

```
>>> re.search("[^0-9a-zA-Z]", "MustafaBaşer")
<SRE_Match object at 0x00A474B0>
```

Gördüğünüz gibi Türkçe harf var diye eşleme yapıldı. Eşleme içine diğer karakterleri teker teker koyabilirsiniz:

```
>>> re.search("[^0-9a-zA-Z ıİşŞöÖçÇüÜğĞ]", "Mustafa Başer")
>>>
```

Karakter eşlemeleri yapareken aşağıdaki özel dizeleri kullanabilirsiniz:

Dize	Anlamı
\d	herhangi bir rakamı eşler, [0,9]
\D	rakam olmayanı eşler, [^0,9]
\s	beyaz boşluğu eşler, [\t\n\r\f\v]
\S	beyaz boşluk olmayanı eşler, [^ \t\n\r\f\v]
\w	sayı ve rakama eşleme yapar, [a-zA-Z0-9_]
\W	sayı ve rakam olmayana eşleme yapar, [^a-zA-Z0-9_]

Örnekler:

```
>>> re.search("\d", "989898")
<SRE_Match object at 0x012BD440>
>>> re.search("\d", "dgdfg")
>>> re.search("\D", "dgdfg")
<SRE_Match object at 0x012C3490>
>>> re.search("\D", "dgdfg9")
<SRE_Match object at 0x012C6CA0>
>>> re.search("\D", "989898")
>>> re.search("\w", "dgdfg9")
<SRE_Match object at 0x012CB7D0>
>>> re.search("\w", "dgdfg9ğü")
<SRE_Match object at 0x012CC3E0>
>>> re.search("\W", "dgdfg9ğü")
<SRE_Match object at 0x012C4820>
>>> re.search("\w", "ğü")
>>> re.search("\W", "ğü")
<SRE_Match object at 0x012D1110>
```

Meta-Karakterler

Yukarıdaki örneklerde kullandığımız [], ^ ve – karakterlerinin özel anlamları vardır. Bu tür karakterlere **meta karakter** denir. Diğer meta karakterler ve anlamları şöyle:

Meta Karakter	Anlamı
.	herhangi bir karakter
^	değil
$	satır sonu
*	önceki karakterden herhangi bir sayıda
+	önceki karakterden bir tane daha
?	önceki karakterden bir tane veya hiç
{	önceki karakterin sayısı
[]	karaker aralığı
\	sonrakinin meta karakter olmadığını belirtir
|	veya
()	desen gruplama

Satır sonundaki bir karaktere eşlemede yapabilirsiniz, bunun için $ meta karakterini kullanalım:

```
>>> re.match("</html>$","</html><!--Bitti-->")
>>> re.match("</html>$","</html>")
<SRE_Match object at 0x00AB2990>
>>> re.match("</html>$","</html>\n")
<SRE_Match object at 0x00AB5AC0>
```

Metin içerisinde herhangi iki desenden birisine eşleme yapmak isterseniz "|" veya meta karakterini kullanınız:

```
>>> re.search("<h1>|<h2>","<h3>Burası üç numaralı başlık</h3>")
>>> re.search("<h1>|<h2>","<h1>Bir numaralı başlık</h1>")
<SRE_Match object at 0x00AE5740>
>>> re.search("<h1>|<h2>","<h2>İki numaralı başlık</h2>")
<SRE_Match object at 0x00AE8FC0>
```

Tekrarlamalar

Metin içerisinde bir deseni ararken bazen bir karakterin belirli bir sayıda tekrarlanıp tekrarlanmadığına bakmak isteyebilirsiniz. Bunun için yukarıda belirtiğimiz meta-karakterleri kullanırsınız. Örneğin "<" ile başlayıp ">" ile biten ve arasında sadece bir karakter bulunan desenleri arayalım: <b>, <i> gibi. Bunun için "." meta karakterini kullanabilirsiniz:

```
>>> re.match("<.>","<i>")
<SRE_Match object at 0x00AB1960>
>>> re.match("<.>","</h1>")
>>> re.match("<.>","<b>")
<SRE_Match object at 0x00AB3AD0>
```

Yukarıda "</h1>"in eşlenmediğini görüyorsunuz. Herhangi bir karakterden herhangi bir sayıda olmasını istiyorsanız bu durumda * meta karakterini kullanacaksınız:

```
>>> re.match("<*>","</h1>")
```

Yine eşleme yapılmadığını gördük. Neden? Çünkü * meta karakteri, kendisinden önce gelen karakterden herhangi bir sayıda olana eşlenir, bu durumda bir bir nokta "." meta karakterini kullanmamız gerekir:

```
>>> re.match("<.*>","</h1>")
<SRE_Match object at 0x00AC47F0>
>>> re.match("<.*>","<tr>")
<SRE_Match object at 0x00AC8050>
>>> re.match("<.*>","</td>")
<SRE_Match object at 0x00ACDF90>
```

Şimdi ben sadece </td>, <h1>, <tr>, </tr> gibi, "<>" arasında iki veya üç tane karakter olan cümleleri eşlemek isterken <table> eşlenmesin istiyorum. "Ne yapmam gerekli?" sorusuna verilecek yanıt meta karakter tablosunda gizli. {} meta karakteri tekrarlama sayısını gösteriyordu:

```
>>> re.match("<.{2,3}>","</h1>")
```

```
<SRE_Match object at 0x00AE0F30>
>>> re.match("<.{2,3}>","<h1>")
<SRE_Match object at 0x00B01A60>
>>> re.match("<.{2,3}>","<b>")
>>> re.match("<.{2,3}>","<table>")
>>> re.match("<.{2,3}>","</td>")
<SRE_Match object at 0x00B0B750>
```

Eşlenen Nesnesinin Özellikleri

Yukarıda, eşleme yapıldığında bir nesne veriliyor dedik. Bu nesnenin çeşitli özellikleri vardır ve çoğu zaman kullanılması kaçınılmazdır. Özellikler şöyle:

Özellik	Amacı
group()	Re ile eşlenen cümleyi çağırır.
start()	Re ile eşlenen cümlenin başladığı konumu verir.
end()	Re ile eşlenen cümlenin bittiği konumu verir.
span()	Re ile eşlenen cümlenin başladığı ve bittiği konumu tüp halinde verir.

Şimdi örnekler görelim. Biz bir html sayfası içerisindeki başlığı arıyor olalım. Html sayfalarında başlık <title> ve </title> tagları arasına yazılır. Sayfamız şöyle olsun:

```
>>> HTML="""
<html>
<head><title>Bu bir deneme sayfasidir</title></head>
<body>
Sayfa burada başlıyor
</body>
</html>
"""
```

Burada Re'miz "**<title>.*</title>**" şeklindedir. Şimdi re'yi yazalım:

```
>>> re.search("<title>.*</title>", HTML)
<SRE_Match object at 0x012A59B0>
>>> re.search("<title>.*</title>", HTML).group()
'<title>Bu bir deneme sayfasidir</title>'
```

```
>>> re.search("<title>.*</title>", HTML).start()
14
>>> re.search("<title>.*</title>", HTML).end()
53
>>> re.search("<title>.*</title>", HTML).span()
(14, 53)
>>> HTML[14:53]
'<title>Bu bir deneme sayfasidir</title>'
```

Harf Duyarlılığına Son

Şimdiye kadar yaptığımız örneklerde hep büyük küçük harf duyarlılığı yapıldı. Özellikle yukarıdaki örnekte bu çok önemli, çünkü html sayfasını yazan kişinin **<title></title>** mı yoksa **<TITLE></TITLE>** mı yazdığını bilemeyiz. Her ikiside html yazım kurallarına göre doğrudur. Bunun için harf duyarlılığını iptal etmek gerekir:

```
>>> re.search("<TITLE>.*</TITLE>", HTML)
>>> re.search("<TITLE>.*</TITLE>", HTML, re.IGNORECASE)
<SRE_Match object at 0x012E0EE0>
```

Aslında aynı sonuca şu şekilde de ulaşabilirsiniz:

```
>>> re.search("<TITLE>.*</TITLE>", HTML.upper())
<SRE_Match object at 0x012DFE50>
```

Re Grupları

Metin içerisinde aradığımız desen birden fazla olabilir, o zaman () meta karakteri ile gruplama yöntemine gideriz. Örneğin bir metin içerisinde tarih yazılı yeri arayalım. Tarih genellikle 09/02/2001 veya 12/10/99 şeklinde yazılır. O halde gruplarımız şöyle olmalıdır:

"(\d{2})/(\d{2})/(\d{2,4})". Örneğin:

```
>>> tarih="Salı 09/02/2001"
>>> re.search("(\d{2})/(\d{2})/(\d{2,4})", tarih)
<SRE_Match object at 0x00ADB8A0>
```

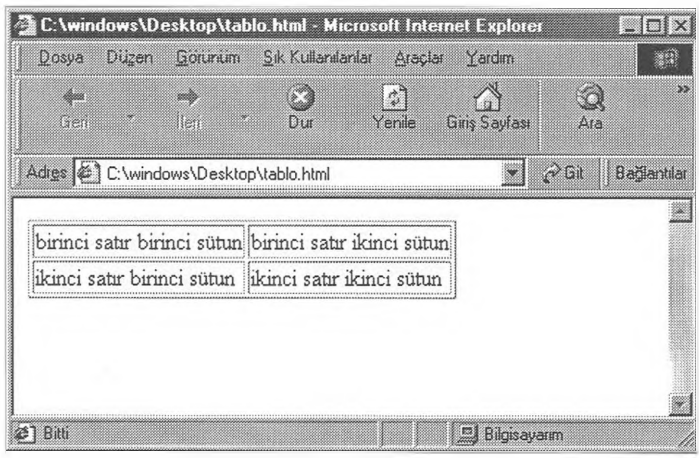
Desenlere Ayırma ve Yer Değiştirme

Re sadece belirli bir desenin metin içerisinde olup olmadığını araştırmaz, aynı zamanda belirli bir desene göre metni parçalayabilir veya belirli bir desen yerine sizin istediğiniz bir deseni metin içerisine yerleştirebilir.

Yer değiştirme işi **sub()** özelliği ile yapılır. Örneğin bir metin içerisindeki tüm beyaz boşlukları yok etmek için şöyle bir yer değiştirme yapılabilir:

```
>>> metin="""Dağ başını duman almış
Gümüş dere durmaz akar
Güneş ufuktan şimdi doğar
"""
>>> print re.sub("\s", "",metin)
DağbaşınıdumanalmışGümüşdereduramzakarGüneşufuktanşimdidoğar
```

Ayırma işlemi daha önceden alışık olduğunuz **split()** özelliği ile gerçekleştirilir. Şimdi taraycıdaki görüntüsü **Şekil-9.1**'deki gibi olan bir tablonun elemanlarını almaya çalışalım.



Şekil-9.1: Elemanları alınacak tablonun tarayıcıdaki görüntüsü

Bu tablonun html metni şu şekildedir:

```
>>> TABLO="""
<table border=1>

  <tr>
       <td>birinci satır birinci sütun</td>
```

```
        <td>birinci satır ikinci sütun</td>
    </tr>

    <tr>
        <td>ikinci satır birinci sütun</td>
        <td>ikinci satır ikinci sütun</td>
    </tr>

</table>
"""
```

İlk olarak metin içerisinde istemediğimiz <table border=1>, </table>, <tr> ve </tr> taglarını bir çırpıda yok edelim:

```
>>> TABLO=re.sub("<.?table.*>|<.?tr>","",TABLO)
```

Şimdi ayırma işlemi yapıp boş elemanları atalım:

```
>>> TABLO=re.split("<.?td>",TABLO)
>>> for i in range(len(TABLO)):
    try:
        if re.match("\s.*",TABLO[i]): TABLO.remove(TABLO[i])
    except:
        pass
```

Eveet, tablo elemanlarını alabiliriz:

```
>>> for i in TABLO: print i

birinci satır birinci sütun
birinci satır ikinci sütun
ikinci satır birinci sütun
ikinci satır ikinci sütun
```

Bu son örneğimiz birçok şeyi içermesi açısından sanıyorum çok faydalı oldu.

Re'leri Derlemek

Eğer bir Re sürekli olarak kullanılacaksa hız kazanmak açısından onu hafızaya derleyip saklamanız ve yeri geldikçe kullanmanız, hızınızı artıracaktır. Derleme işlemi çok kolay bir şekilde yapılır:

```
>>> RE=re.compile("[^0-9a-zA-Z ıİşŞöÖçÇüÜğĞ]")
>>> RE.search('Mustafa BAŞER')
>>> RE.search('Mustafa BAŞER mi geldi?')
<SRE_Match object at 0x00A983F0>
>>> RE.search('Heey uyan artık!')
<SRE_Match object at 0x00AAB530>
```

Sınıflar



Nesne tabanlı programcılığa hoş geldiniz. Şimdiye kadar hep nesneleri ve onların özelliklerini kullandık. Bu nesnelerin nasıl oluşturulduğunu incelemedik. Bu bölümde sınıfları dolayısıyla gerçek anlamda nesneleri inceleyeceğiz.

Sınıfları birçok değişken ve fonksiyondan oluşan büyük fonksiyonlar gibi düşünebilirsiniz. Ancak bir fonksiyon içerisindeki değişkenlere dışardan ulaşamaz ve değişiklik yapamazsınız, oysaki sınıflarda fonksiyonlara ulaşabilir ve değişkenlerin değerlerini alabilir veya değiştirebilirsiniz.

Aslında sınıfları kullanmadan da çok iyi programlar yazabilirsiniz, ancak birçok zorlukla karşı karşıya gelirsiniz. Program içerisindeki değişkenler çeşitli fonksiyonlar içerisinden değiştirilecekse bu durumda her yerde global yazmakla uğraşırsınız. Tam anlamıyla nesne tabanlı program yazamamış olursunuz. Düşünün hafızada duran bir dosya oluşturmak istiyorsunuz. Bir nesneye ihtiyacınız var. Ona Write ile yazıp read ile okumanız gerekli. Neden mi, çünkü bazı hazır fonksiyonlar dosyalara yazar ve onları okurlar, oysaki dosyaya yazma ve okuma zahmetli bir iştir ve sistemi çok yorar. Kısacası iyi bir Python programı sınıfsız olamaz. Zaten ilerde mecburen sınıf programcılığı yapmak zorunda kalacağız.

Sınıfları anlatmanın ne kadar zor olduğunu biliyoruz. Aslında öğrenmesi o kadar zor değil, fakat anlatması çok zor. Mümkün olduğu kadar çok örnek vererek ve kolaydan zora doğru giderek anlayabilmeniz için geniş açıklamalara yer vereceğiz. Bölümü bitirdiğinizde taşlar kafanızda yerli yerine oturmadıysa, yeniden okumak yerine öğrenmeye biraz ara verin ve eski bilgilerin kafanızda çimlenmesini bekleyin. Daha sonra tekrar okuyarak bilgilerin pekişmesini sağlayın.

Sınıfların Oluşturulması

Python'da sınıflar **class** hazır fonksiyonu ile oluşturulur. Yazılımı şöyledir:

class sinif_ismi:
    Sınıfa ait
    fonksiyonlar

Sınıflarla ilgili şu kısa notları verelim:

- Bir sınıf çağırıldığında onda ilk işletilen fonksiyon **__init__**() özel fonksiyonudur. Bir sınıfın nesneleri onu belirten bir taşıyıcı ile birbirine bağlanır (**self**).
- Dışardan ulaşılmasını istediğiniz her nesne **self**'e bağlanmalıdır.
- Sınıf içersinde her yerden ulaşılacak ve değiştirilecek fonksiyon ve değişkenler **self**'e bağlanmalıdır. Bu bağlanma, değişkenlerde **self**'in bir özelliği gibi yapılırken, fonksiyonlarda ilk argüman olarak verilir.

Şimdi hemen bir sınıf oluşturalım. Bu sınıfta sadece iki tane değişken olsun:

```
>>> class SINIF:
        def __init__(self):
                self.no=541407
                self.isim="Mustafa BAŞER"
```

Yıkardaki tanımlanan sınıfı ilk çağırdığımızda **__init__**() fonksiyonu işletilecek, bundan sonraki çağrımlarda işletilmeyecektir. Bu nedenle bir sınıfta sadece bir kere atama yapılacak olan değişkenler burada tanımlanır. Bu sınıfa ait herhangi bir fonksiyon program akışı içerisinde çağrıldığında bu fonksiyon işletilmez. Bu tanımlanan fonksiyonun bu sınıfa ait olduğunu, sınıf içerisinden ve dışarısından erişilebileceğini **self** argümanı ile belirlemiş olduk. Benzer şekilde **no** ve **isim** değişkenlerini de **self**'in bir özelliği gibi (aslında öyle, çünkü **self** sınıfı ifade ediyor ve **no** ile **isim** değişkenleri de sınıfın bir özelliği) yaptık.

Sınıflar çağırılırken önce bir nesneye sanki bir fonksiyonmuş gibi atanır:

```
>>> S=SINIF()
```

Şimdi **S** nesnesinin özelliklerine bakalım:

```
>>> dir(S)
['__doc__', '__init__', '__module__', 'isim', 'no']
```

Gördüğünüz gibi **no** ve **isim** değişkenleri erişilmeyi bekliyor:

```
>>> print S.no
541407
>>> print S.isim
Mustafa BAŞER
```

Şimdi **S** nesnesindeki bu değişkenlerin değerlerini değiştirelim:

```
>>> S.no=87656
>>> S.isim="Fatih BAŞER"
>>> print S.no, S.isim
87656 Fatih BAŞER
```

SINIF sınıfını başka bir nesneye de atayabilirsiniz. Her iki nesnenin içeriği farklı olabilir:

```
>>> S2=SINIF()
>>> print S2.no, S2.isim
541407 Mustafa BAŞER
>>> S2.no=189
>>> print S2.no, S2.isim
189 Mustafa BAŞER
>>> print S.no, S.isim
87656 Fatih BAŞER
```

Gördüğünüz gibi **S2** nesnesi **SINIF** cinsinden fakat yine **SINIF** cinsinden olan **S** nesnesi ile farklıdır. Birinde yapılan bir değişiklik diğerini hiç etkilemiyor.

Sınıflara Veri Aktarımı

Sınıflara tıpkı fonksiyonlar gibi veri aktarabilirsiniz. Veri aktarma işlemi **__init__()** özel fonksiyonu ile yapılabilir. Bu fonksiyona gönderilen bilgiler, içerden ve dışardan ulaşılsın diye **self**'in birer özelliği gibi belirlenir. Şimdi yukardaki sınıfı biraz değiştirelim:

```
>>> class SINIF:
        def __init__(self, noGelen, isimGelen):
            self.no=noGelen
            self.isim=isimGelen
```

Bu sınıfta, **__init__()** özel fonksiyonuna gelen **noGelen** ve **isimGelen** , sınıfa ait **no** ve **isim** değişkenlerine sırası ile atanıyor. Örneğin:

```
>>> S=SINIF(55,"Fatih Başer")
>>> print S.no, S.isim
55 Fatih Başer
```

Yukardakine benzer şekilde aynı **SINIF** sınıfını başka bir nesneya atayıp bağımsız şekilde kullanabiliriz:

```
>>> S2=SINIF(56,"Melike Başer")
>>> print S.no, S.isim
55 Fatih Başer
>>> print S2.no, S2.isim
56 Melike Başer
```

Veri aktarımını kendi yazacağınız bir fonksiyon ile de yapabilirsiniz. Bunun örneğini ilerde göreceğiz.

Sınıf İçi Fonksiyonların Tanımlanması

Bir sınıfın içerisinde fonksiyon tanımlanırken, sanki normal bir fonksiyon tanımlarmış gibi yapılır. Tek farkı, fonksiyonun ilk argümanının mutlaka **self** olmasıdır. Aşağıda **SINIF** sınıfındaki değerleri ekrana bastırmaya yarayan **yaz()** fonksiyonunu görüyorsunuz.

```
>>> class SINIF:
        def __init__(self, noGelen, isimGelen):
            self.no=noGelen
            self.isim=isimGelen
        def yaz(self):
            print self.no, self.isim
```

Sınıf içerisindeki fonksiyonların nasıl çağrıldığını önceki bölümlerde gördük. Bir sınıfın fonksiyonu, atama yapılmış nesnenin özelliği olarak çağırılır. Aşağıda önce **SINIF**'ı **S** nesnesine atadık sonra da içerisindeki **yaz()** fonksiyonunu çağırdık:

```
>>> S=SINIF(99, "Hasan Yalçınkaya")
>>> S.yaz()
99 Hasan Yalçınkaya
```

Sınıf içerisindeki fonksiyonlar, yine içerdeki başka bir fonksiyon tarafından çağrılabilir. Çağırma işlemi **self**'in bir özelliğiymiş gibi yapılır. Şimdi aşağıdaki örneğe bakalım:

```
>>> class SINIF:
    def __init__(self, noGelen, isimGelen):
        self.no=noGelen
        self.isim=isimGelen
    def yaz(self):
        print self.no, self.isim
    def deneme(self):
        print "Şimdi sınıf içerisindeki yaz() fonksiyonu \
çağırılacak"
        self.yaz()
```

Burada **deneme()** fonksiyonunu yazdık. Çağırdığımızda önce ekrana bir bilgi yazıyor ve sonra da **yaz()** fonksiyonunu çağırıyor. Aşağıda kullanımını görelim:

```
>>> S.deneme()
Şimdi sınıf içerisindeki yaz() fonksiyonu çağırılacak
45 Mustafa Akgün
```

Banka Uygulaması

Sınıfları anlatmanın çok kolay olmadığını bölümün başında belirtmiştik. Bu konuyu en iyi açıklayan örneklerden birisini Addison Wesley yayınları arasında Alan Gauld tarafından yazılan *Learn to Program Using Python* isimli kitaptan çok az değişiklik yaparak buraya aldık.

Aşağıda basit bir banka uygulaması yazacağız. Bu uygulamada temel işlemlerden şunlar yapılacak:

- Hesap açma
- Hesaba para yatırma
- Hesaptan para çekme
- Bir hesaptan başka bir hesaba para tarnsferi

Elbette bankacılık programları çok karmaşık bir yapıya sahiptir. Biz burada sınıflar kullanılarak bu işlemlerin ne kadar kolay bir hale gelebileceğini açıklamaya çalışıyoruz. Yukarıdaki işleri yapacak olan sınıfı şöyle yazabiliriz:

```
class BANKAHESABI:
    def __init__(self, ilkMiktar=0):
        self.kalanPara=ilkMiktar
        print "Yeni hesap %d TL ile açılmıştır" % ilkMiktar

    def paraYatirma(self, yatanMiktar):
        self.kalanPara=self.kalanPara+yatanMiktar

    def paraCekme(self, cekilenMiktar):
        self.kalanPara=self.kalanPara-cekilenMiktar

    def kalanPara(self):
        return self.kalanPara

    def havale(self, miktar, kime):
        self.paraCekme(miktar)
        kime.paraYatirma(miktar)
```

Bu kodu **bankahesabi.py** olarak kaydettikten sonra bir uygulama yapalım. Önce 65.000 TL ile Fatih ve 25.000 TL ile de Melike'ye ait banka hesapları açalım:

```
>>> from bankahesabi import BANKAHESABI
>>> fatih=BANKAHESABI(65000)
Yeni hesap 65000 TL ile açılmıştır
>>> melike=BANKAHESABI(25000)
Yeni hesap 25000 TL ile açılmıştır
```

Şimdi **Fatih**'e 18.000 TL para yatıralım ve hesabında kaç para kalmış bakalım:

```
>>> fatih.paraYatirma(18000)
>>> fatih.kalanPara
83000
```

Fatih'ten **Melike**'ye bir havale yapalım, havale miktarı 13.000 TL olsun:

```
>>> fatih.havale(13000, melike)
>>> melike.kalanPara
38000
>>> fatih.kalanPara
70000
```

Gördüğünüz gibi, çok karmaşık gibi görünen işlemler sınıflar ile oldukça kolay bir şekilde yapılabiliyor.

Miras Alma

Eğer bir programlama dilinde sınıflar varsa mutlaka miras alma da vardır. aksi halde sınıfların kullanılması anlamsızlaşır. Miras alma bir sınıfın tüm nesnelerini başka bir sınıfın alması ya da kullanmasıdır. Miras alınan sınıfa ebeveyn veya temel denilir. Özellikle modüler programlarda sıkça kullanılan miras almanın kullanımı şu şekildedir:

class *sinifIsmi(ebeveyn):*
sinif içi
tanımlamalar

Bunu örnekleyelim:

```
>>> class A:
        def __init__(self):
            self.liste=[]
        def ekle(self,gelen):
            self.liste.append(gelen)

>>> class B(A):
        def yaz(self):
            print self.liste
```

Yukarıda **A** ve **B** isimleri ile iki tane sınıf tanımladık. **A** sınıfı **B** sınıfı için ebeveyn oldu. Yani **A** sınıfına ait tüm erişilebilir nesneleri **B** içerisinde kullanabiliriz:

```
>>> S=B()
>>> S.ekle("mbaser")
>>> S.ekle("mmakgun")
>>> S.yaz()
['mbaser', 'mmakgun']
```

Önce **B** sınıfını **S** nesnesine atadık. Sonra **ekle** fonksiyonunun sanki **B** sınıfının bir özelliğiymiş gibi kullandık. Dikkat ederseniz, **B** sınıfındaki **ekle()** fonksiyonunda kullanılan liste değişkeni kendi değişkeni değil, ebeveyninden aldığı değişkendir. Bu değişkene istediğimiz değerleri atadık. Son olarak **B** sınıfına ait **yaz()** fonksiyonunu kullandık. Bu fonksiyon içersinde kullanılan liste değişkeni yine ebeveynden gelen değişkendir.

Python çoklu miras almaya da izin verir. Bunun kullanımı ve tanımlaması aynen tekli miras gibidir. Tek farkı ebeveyn sınıfların virgül ile birbirinden ayrılmasıdır:

class *sinifIsmi(ebeveyn1, ebeveyn2, ebeveny3 ....):*
sinif içi
tanımlamalar

Burada yeni oluşturulan sınıf tüm ebeveynlerdeki erişilebilir değişken ve fonksiyonlara ulaşabilirken ebeveynler kendileri aralarında ulaşamazlar (Elbette bir ebeveyn başka bir ebeveynden miras almamışsa).

Operatör Yüklemesi

Birçok programlama dili ön tanımlı operatörlerin aşırı yüklenmesine olanak verir. Aşırı yüklemeden kasıt ön tanımlı operatörlerin (toplama, çıkarma, bölme...) kullanıcı tarafından tanımlanan nesneler üzerindeki etkisinin yine kullanıcı tarafından belirlenmesidir. Konumuzu açıklamak için Python tarafından kullanılan karmaşık sayıları kendimiz düzenleyelim. Bildiğiniz gibi karmaşık sayıların bir gerçek bir de sanal kısmı vardır. Aşağıdaki karmaşık sayıya bakalım:

```
>>> c=4+3j
>>> c.real
4.0
>>> c.imag
3.0
```

Şimdi böyle bir sayıyı kendi tanımlayacağımız bir nesne ile oluşturalım:

```
>>> class karmasik:
        def __init__(self,g=0,s=0):
            self.gercek=g
            self.sanal=s

>>> c1=karmasik(3,4)
>>> c2=karmasik(2,-5)
>>> print c1.gercek, c1.sanal
3 4
>>> print c2.gercek, c2.sanal
2 -5
```

Gördüğünüz gibi tıpkı Python karmaşık sayılarına benzer bir yapı oluşturduk ve **c1** ile **c2** isimlerinde iki tane karmaşık sayı yaptık. Karmaşık sayılarda toplama işlemi gerçek ve sanal kısımların birbirleri ile toplanmasından elde edilir. Şimdi kendi kendimize nasıl yaptığımıza bakalım:

```
>>> toplam=karmasik(c1.gercek+c2.gercek, c1.sanal+c2.sanal)
>>> print toplam.gercek, toplam.sanal
5 -1
```

Şimdi aynı işlemleri + (artı) operatörünü aşırı yükleyerek kendi karmaşık sayılar nesnemize etkiletelim. Bunun için **karmasik** sınıfımızı değişikliğe uğratarak **__add__()** özel fonksiyonunu ekleyeceğiz:

```
>>> class karmasik:
    def __init__(self,g=0,s=0):
        self.gercek=g
        self.sanal=s
    def __add__(self, diger):
        return karmasik(self.gercek+self.gercek,
diger.sanal+diger.sanal)

>>> c1=karmasik(3,4)
>>> c2=karmasik(2,-5)
>>> toplam=c1+c2
>>> print toplam.gercek, toplam.sanal
5 -1
```

Benzer şekilde - (eksi) operatörünü aşırı yüklemek için **__coerce__()**, * operatörünü aşırı yüklemek için **__mult__()**, / operatörünü aşırı yüklemek için ise **__div__()** özel fonksiyonları kullanılabilir. Bu fonksiyonların alacağı ilk parametrenin **self**, diğer parametrenin ise herhangi bir şekilde isimlendirileceğini unutmayın.

Dosya, Klasör ve Süreç İşlemleri

11

Bir programcının eninde sonunda kullanmak zorunda kaldığı dosya, klasör ve süreçlerin denetlenmesi ve yeniden yapılandırılması gibi konuları bu bölümde inceleyeceğiz. Çeşitli dosya ve klasör işlemleri ile bir sürecin açılması, ona veri aktarımı ve sonucun okunması gibi işlemleri yapacağız. Bunları yapmak için Python'da birçok hazır kütüphane (modül) bulunuyor. Alt düzey programcılıkla hiç uğraşmadan tüm bu işleri rahatlıkla yapabileceksiniz. İnceleyeceğimiz modüller arasında **os**, **fcntl**, **shutils** ve **sys** bulunuyor. Bu modüllerde bulunan tüm fonksiyonlara bakmamız olanaksız, incelemediklerimize *Python Library Rerefence*'den okuyabilirsiniz.

Bu bölümü baştan sona kadar okumak yerine gerektiğinde referans olarak kullanabilirsiniz. Elbette neler olup olmadığını şöylece bir gözden geçirmeniz gerekiyor.

os Modülü

os modülü kısaca işletim sistemi ile alakalı olarak yapılacak işleri belirtir. İşletim sistemleri arasında kullanım bakımından ufak tefek farklar gösterse de burada ele alacağımız fonksiyonlar hemen hemen tüm işletim sistemleri üzerinde geçerlidir.

os.name

İşletim sisteminizin tipini belirtir. Bu tip şunlardan birisi olabilir:

```
'posix', 'nt', 'dos', 'mac', 'os2', 'ce', 'java'.
```

Windows makinalar üzerinde verilecek sonucun "nt" olduğu muhakkaktır. Windows98 üzerinde bu özelliği çağırdığımızda **Şekil-11.1**'deki gibi bir ekran ile karşılaşmalısınız. UN*X tipi işletim sistemlerinde bu sonucun "posix" olması gerekiyor

```
>>> import os
>>> os.name
'nt'
```



Şekil-11.1: İşletim sisteminizi öğrenin: os.name

os.environ

Bulunduğunuz ortamdaki çevre değişkenlerinin alınması ve yeniden yapılandırılması için kullanılır. Özellikle UN*X kullanıcıları bu özelliğini oldukça sık kullanırlar. Verilen sonuç çevre değişkenlerinin adlarından oluşan bir sözlüktür:

```
>>> for e in os.environ.keys():
        print e, "-->",os.environ[e]

TMP --> C:\WINDOWS\TEMP
WINDIR --> C:\WINDOWS
PROMPT --> $p$g
COMSPEC --> C:\WINDOWS\COMMAND.COM
```

```
TEMP --> C:\WINDOWS\TEMP
CMDLINE --> WIN
BLASTER --> A220 I7 D1 H7 P330 T6
SBPCI --> C:\SBPCI
TCL_LIBRARY --> C:\PYTHON22\tcl\tcl8.3
WINBOOTDIR --> C:\WINDOWS
TK_LIBRARY --> C:\PYTHON22\tcl\tk8.3
PATH --> C:\PROGRA~1\ABRIAM~1\PERL\BIN;C:\WINDOWS;C:\WINDOWS\COM-
MAND;C:\BITWARE\;C:\PYTHON22;C:\PROGRAM FILES\MTS
```

os.system(), os.popen(), os.popen2()

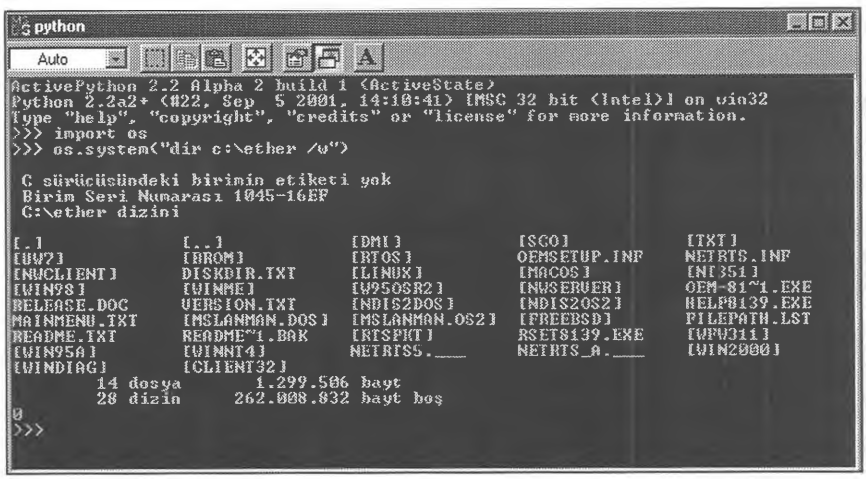
UN*X'in komut satırındaki gücü tartışılmazdır. Hemen her işinizi komut satırında bulacağınız bir program ile, kullanımı biraz sıkıcı da olsa yapabilirsiniz. Bazı programlar kendilerine iletilecek verileri argüman olarak alırken bazıları da standart girdiden okurlar. Eğer programın sonucunda bir şeyler bekliyorsanız bu durumda o programın standart çıktısını okumanız gerekecek. Bu işlemleri yapabileceğiniz os fonksiyonları **system()**, **popen()** ve **popen2**'dir.

system() fonksiyonu, açılan alt sürece bir boru açmaz, dolayısı ile ne sürecin standart girdisine yazabilir, ne de okuma yapabilirsiniz. Amacınız sadece bir programı çalıştırmaksa bu fonksiyonu kullanabilirsiniz. İyi yazılmış bir program çıkış durumunda mutlaka bir değer döndürür. Eğer program hatasız bir şekilde çalışmış ve sonlanmış ise bu sıfır (0) olur. **system()** fonksiyonu alt sürecin çalışıp bitmesini bekler ve alt sürecin çıkış değerini döndürür. Şimdi Windows altında bulunan **notepad.exe** programını çalıştırıp sonucuna bakalım:

```
>>> import os
>>> os.system("notepad")
>>> os.system("notepad.exe")
0
```

Yukarıdaki satırları işlettiğinizde **notepad.exe** çalışacak, Windows GUI ana döngüsüne girdiği zaman programın sonuna gelinecek ve alt sürecin çalıştırma işlemi sonlanmış olacaktır. Eğer bir hata vermeden çalışmışsa, sonuç sıfır olarak dönecektir. Eğer çalıştırdığınız program bir DOS programını ça-

lıştırmış olsaydı, program standart çıktıya bir şeyler yazmış olsaydı ve biz etkileşimli Python yorumlayıcı kullanıyorsak, sonuç ekranımıza yazılacaktı. Alınan pencere örneğini **Şekil-11.2**'de görüyorsunuz.



Şekil-11.2: Dir DOS komutunun etkileşimli Python yorumlayıcısı içerisinden çalıştırılması.

Bu komut bir Python programı içerisinden çalıştırılmış olsaydı, sonucu göremeyecektik. Çalıştırılan bir programın sonucunu okuyabilmek için **popen()** veya **popen2()** kullanmamız gerekirdi. Şimdi **dir** komutunun çıktısını okuyalım:

```
>>> p=os.popen("dir c:\ether")
>>> dosyalar=p.readlines()
>>> print dosyalar[16]
DISKDIR  TXT            948  14/08/00  19:00 diskdir.txt
```

Burada "**dir**" DOS komutuna bir boru açtık. Açtığımız **p** borusu sayesinde, sanki bir dosyadan okuyormuş gibi komutun çıktısını okuduk. Okuduğumuz değerler bir liste halinde **dosyalar** değişkenine atandı. Buradan istediğiniz satırı çağırıp inceleyebilirsiniz.

dir komutuna bir bilgi veriyoruz, bu bilgi listelenmesini istediğimiz patika. Bu bilgiyi komut satırından argüman olarak verdik. Bazı programlar ise ve-

rileri standart girdiden okurlar. Buna bir örnek, hem Windows hem de UN*X platformunda bulunan **sort** komutudur. Bu komut kendisine standart girdiden verilen satırları sıralayarak ya ekrana ya da yönlendirme yaptığınız herhangi bir aygıta (dosyaya) yazabilir. Şimdi bu işlemleri yapalım:

```
>>> p=os.popen("sort >c:\\s.txt","w")
>>> p.write("Mustafa\n")
>>> p.write("Fatih\n")
>>> p.write("Ali\n")
>>> p.write("Dilek\n")
>>> p.close()
```

Yukardaki programda ilk olarak bir süreç başlatıyoruz. Sanki bir dosya açarmış gibi süreci yazılabilir halde açıyoruz ("w" modu ile). Daha sonra **p** sürecine sanki bir doyaya yazıyormuş gibi satırlar yazıyoruz. Bu yazılanlar **sort** programının standart girdisine veriliyor. Yine tıpkı bir dosyayı kapatıyormuş gibi, standart girdiye yazma işleminin son bulduğunu **sort** programına iletiyoruz. **sort** programı sıralamayı yaptıktan sonra **c:\s.txt** dosyasına yazıyor. Şimdi bu dosyayı okuyup içinde ne var ne yok bakalım:

```
>>> print open("c:\\s.txt").read()
Ali
Dilek
Fatih
Mustafa
```

Gördüğünüz gibi işlemler doğru bir şekilde yapılmış. **popen()** bir süreci ya okuma ya da yazma modunda açabilir. "Peki biz **sort** programının girdisine birşeyler yazıp aynı zamanda çıktısını da okuyabilir miyiz?" sorusuna şimdi yanıt bulalım:

```
>>> (pi, po)=os.popen2("sort")
>>> pi.write("Fatih\n")
>>> pi.write("Fatih\n")
>>> pi.write("Melike\n")
>>> pi.write("Dilek\n")
```

Python

```
>>> pi.close()
>>> sonuc=po.read()
>>> print sonuc
Dilek
Fatih
Fatih
Melike
```

Bu programda olup bitenleri şöyle özetleyebilirz: **popen2()** bir alt süreci başlatırken standart girdi ve çıktısını birer nesneye bağlayarak bir tüp haline getirir. Daha sonra siz tüpün içerisindeki nesneleri kullanarak açılan sürece yazabilir ve sürecin çıktısını okuyabilirsiniz. Biz bu nesneleri **pi** ve **po** olarak tanımladık. Yukarda gördüğünüz gibi pi nesnesine 4 satır yazdık. Bu satırların tamamı sort programına gönderildi ve **pi.close()** ile süreci sonlandırdık. Daha sonra **po.read()** ile çıktıyı bir değişkene atadık. **popen2()** ile aşılan süreçlere aynı anda yazabilir ve okuyabilirsiniz. Okuma işlemi yazma nesnesini kapattıktan sonra olacak diye bir şey yok. Burada sort programı yazma işlemi son bulduğunda standart çıktıya veri yazıyordu, onun için yazma nesnesini kapattıktan sonra okuduk.

Alt süreci başlatacağınız programın standart hatasını da okumak istiyorsanız o zaman **popen3()** kullanmanız gerekiyor. Bu fonksiyon sırasıyla **stdin**, **stdout** ve **stderr** dosya nesnelerini verecektir.

os.chmod(), os.chown

chmod() verilen patikanının özelliklerini değiştirir. Genelde UN*X kullanıcılarının işine yarar.

```
os.chmod("/tmp/myfile",0777)
```

chown() sadece UN*X için vardır ve verilen patikanın kullanıcı ve grup id'sini değiştirir (patika, uid, gid):

```
os.chown("/tmp/myfile", 500, 500)
```

os.listdir()

Bir klasör içerisindeki dosya ve klasörlerin listesini verir. Diğer programlama dillerine göre oldukça kolay kullanımı olan **listdir**(), sistem programcılarının en çok rağbet ettiği fonksiyondur:

```
>>> for d in os.listdir("C:\\Program Files\\Teletext"):
        print d

DeIsL1.isu
CTSERVER.INI
CTSERVER.EXE
CW.INI
DEVICES.INI
SERVICES.TXT
CTSDDE.INI
CTVIEWER.INI
TTX.EXE
TTX.HLP
ctserver.log
```

os.link(), os.symlink(), os.unlink()

link() ve **symlink**(), UN*X dosyal sisteminde sabit link ve sembolik link yapmak için kullnılır. Önce kaynak patikası sonra link yapılacak patika yazılır:

```
>>> os.symlink("/tmp", "/root/tmp")
>>> os.link("/tmp/myfile", "/home/mbaser/myfile")
```

Sabit link yapılacak kaynak ile patikanın aynı fiziksel aygıt üzerinde olması gerektiğini unutmayın. Önceden oluşturulmuş bir linki kaldırmak için **unlink**() fonksiyonunu kullanabilirsiniz. Eğer unlink yaptığınız dosya ise ve sistemde başka sabit link yoksa, dosyanın silineceğini hatırlatalım:

```
os.unlink("/home/mbaser/myfile")
```

os.remove(), os.rmdir()

Dosya ve klasörleri silmek için kullanabilirsiniz:

```
>>> os.remove("c:\\abi\vendor.doc")
>>> os.rmdir("c:\\abi")
```

os.mkdir()

Klasör oluşturan fonksiyondur:

```
>>> os.mkdir("c:\\a")
```

os.rename()

Klasör veya dosyanın ismini yenisi ile değiştirir:

```
>>> os.rename("c:\mbaser", "c:\\mustafa")
```

Eğer kaynak başka bir patika olursa, ismi değiştirmekle kalınmayıp, oraya taşıma işlemi yapılmış olur. Örneğin **c:\log.txt** dosyasını **d:\LOG** klasörüne taşımak için:

```
>>> os.rename("c:\log.txt", "d:\\LOG\\log.txt")
```

os.chdir(), os.getcwd()

Bulunduğunuz yerden başka bir klasöre geçiş yapmak istiyorsanız **chdir()** fonksiyonunu şu anda çalıştığınız klasörü almak için **getcwd()** foknkiyonunu kullanabilirsiniz:

```
>>> os.getcwd()
'C:\\Python22\\Tools\\idle'
>>> os.chdir("C:\\Windows\Desktop")
>>> os.getcwd()
'C:\\Windows\\Desktop'
```

os.stat()

Bir dosya ve kalsörle ilgili tüm özellikleri bu fonksiyon ile alabilirsiniz. Bu fonksiyon çalıştırıldığında 10 adet tamsayı değer döndürür. Bunlar sıra ile:

mod, inode, aygıt, linksayısı, uid, gid, boyutu, erişme zamanı, değiştirme zamanı, oluşturma zamanı

olarak verilir. Windowsda bazı alanlar ön tanımlı ve anlamsız bilgilerle doludur:

```
>>> stat=os.stat("c:\\tmp")
>>> stat
(16895, 0, 2, 1, 0, 0, 0, 1008108000, 982394126, 1008171780)
```

Buradaki ilk sayı Windows'larda klasör ve salt okunurlukla ilgilidir. Diğer değerleri deneyerek bulabilirsiniz. UN*X'lerde ise onluk sistemde dosya modunu göstermektedir. İkinci sayı UN*X makinalarda inod'u gösterirken Windowslarda anlamı yoktur. Üçüncü sayı ise UN*X'lerde bu dosyaya olan sabit link sayısını vermektedir. Dört ve beşinci sayılar kullanıcı ve grubun sayısal değerini, altıncı sayı dosyanın boyutunu verirken son üç sayı sırası ile dosyaya (ya da klasöre) en son erişilen zamanı, en son değişiklik yapılan zamanı ve dosyanın oluşturulduğu zamanı saniye cinsinden vermektedir.

os.path

Os.path'in çok çeşili özellikleri vardır. Bir patika ile ilgili bilmek istediğiniz her şeyi verir. Şimdi çok kullanılan iki özelliğine bakalım:

```
>>> os.path.exists("c:\\tmp")
1
>>> os.path.isdir("c:\\tmp")
1
```

Yukarıdaki ilk fonksiyon, bir patikanın bulunup bulunmadığını denetler, bulunuyorsa 1 yoksa 0 değerini döndürür. İkinci fonksiyon ise verilen patikanın bir klasör olup olmadığını sorar. klasör ise 1, dosya ise 0 değeri döndürülür.

Şimdi **os.listdir()**, **os.path.isdir()** ve **time.ctime()** fonksiyonlarını kullanarak bir örnek bir program yazalım:

program11-01.py

```
#!c:/python22/python.exe

import os, time

toplamBoyut=0
dosyalar=os.listdir("c:\\tmp")
dosyalar.sort()

print "c:\\tmp Klasörünün içeriği:\n"

for dosya in dosyalar:
        tip="dosya"
        if os.path.isdir("c:\\tmp\\%s" % dosya):
                tip="klasör"
        stat=os.stat("c:\\tmp\\%s" % dosya)
        print "%s%s\t%s\t%d" % (dosya.ljust(15),
                               tip,
                               time.ctime(stat[8]),
                               stat[6])

        toplamBoyut=toplamBoyut+stat[6]

print
print "Bu klasörde toplam %d nesne var." % len(dosyalar)
print "Toplam kullanılan alan %d byte'dır." % toplamBoyut
```

Bu programı çalıştırdığımızda aşağıdaki sonuca ulaşıyoruz:

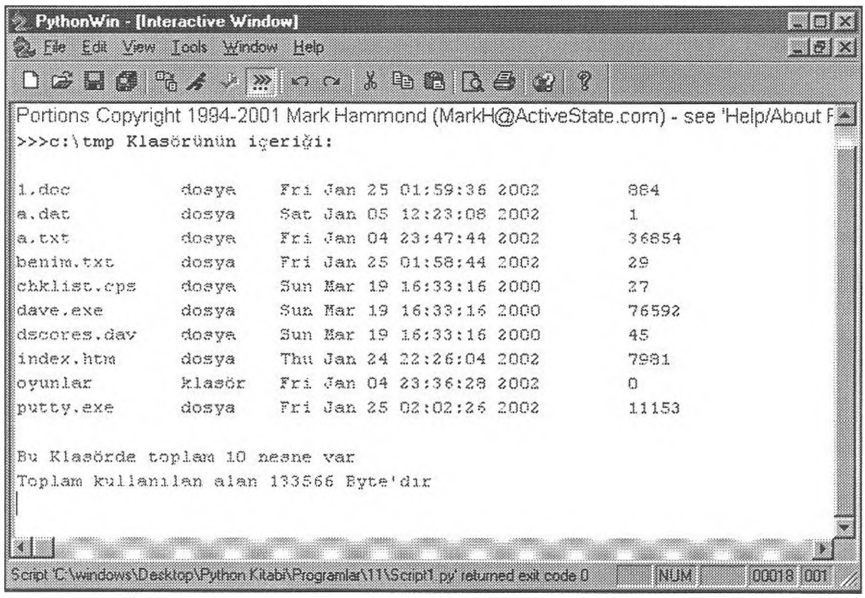
```
c:\tmp klasörünün içeriği:

a.txt           dosya Fri Jan 04 23:47:44 2002     36854
chklist.cps     dosya  Sun Mar 19 16:33:16 2000     27
dave.exe        dosya  Sun Mar 19 16:33:16 2000     76592
```

```
dscores.dav     dosya  Sun Mar 19 16:33:16 2000     45
oyunlar          klasörFri Jan 04 23:36:28 2002     0

Bu klasörde toplam 5 nesne var.
Toplam kullanılan alan 113518 byte'dır.
```

Programın çıktısını aynı zamanda **Şekil-11.3**'de görebilirsiniz.



Şekil-11.3: Bir klasörün içeriğini tam anlamı ile sunan programın örnek çıktısı

Bir dosyanın boyutunu bulmak için **os.path.getsize()** kullanılabilir:

```
>>> os.path.getsize("c:\\PythonProgramlari\\a.py")
289
```

Bir klasörün içeriğini glob modülüyle de alabilirsiniz. Bu modül bildik desenleri kullanmanıza da izin verir:

```
>>> import glob
>>> glob.glob("c:\\tmp\*")
```

```
['c:\\tmp\\a.txt',  'c:\\tmp\\dave.exe',  'c:\\tmp\\oyunlar',
'c:\\tmp\\chklist.cps', 'c:\\tmp\\dscores.dav']
```

Yukarıdaki örnekte **c:\tmp** klasörün içindeki tüm nesneleri * ile istedik. Sonuca dikkat ederseniz elemanların tam patikası var. Eğer sadece .txt dosyalarını istemiş olsaydık:

```
>>> glob.glob("c:\\*.txt")
['c:\\AILog.txt', 'c:\\BOOTLOG.TXT', 'c:\\detlog.txt', 'c:\\net-
log.txt', 'c:\\products.txt', 'c:\\setuplog.txt']
```

Ya da **f** ile başlayan tüm **doc** dosyaları için glob.glob("c:\\mydoc\f*.doc") şeklinde yazabilirdik.

shutil.copytree(), shutil.rmtree()

Daha önce kopyalama ve silme işlemlerinde tek bir dosya veya boş klasörü kopyalayıp silmiştik. Eğer bir klasörü tüm alt klasör ve dosyaları ile başka bir yere kopyalamak veya silmek isterseniz, bu işlemleri **shutil** modülü ile yapmalısınız. Şimdi bir örnek verelim:

```
>>> shutil.copytree("c:\\tmp", "c:\\windows\\desktop\mytmp")
>>> shutil.rmtree("c:\\windows\\desktop\mytmp")
```

Dosya Kilitleme

Dosya kilitleme genellikle çok kullanıcılı ortamlarda bir dosyanın başka bir süreç ya da kişi tarafından ne şekilde açılabileceğini veya açılmayacağını belirler. Yazdığınız program birden çok kişiye hizmet verecekse ve aynı dosya sürekli açılıp kapatılıyorsa, veri kaybını önlemek ve dosyanın sağlığını korumak için mutlaka o dosyayı kilitlemeniz gerekir. Aksi halde bir kullanıcının dosyaya yazdığı bilgilerin bir kısmı yok olabilir veya dosya tamamen sıfırlanabilir.

UN*X işletim sisteminde dosya kilitleme oldukça kolay olmasına rağmen, bu iş Win32'de çok zahmetlidir. Windows/DOS paltformunda dosya kilitlemenin iki yolu vardır: Birisi msvcrt, diğeri ise Win32 API'larını Python içerisinde çağırmaktır. İkinci yol ile dosya kilitlemeyle ilgili örnek uygulamayı *Win32 Extensions Reference*'de bulunan *Windows NT Files – Locking* başlığı altında bulabilirsiniz. Win32 API'larını Python içerisinden kullanmak bu kitabın amaçları dışına çıktığından Windows üzerindeki bu kilitle-

meyi anlatmayacağız. Önce UN*X platformunda, sonra da Windows/DOS üzerinde msvcrt ile dosya kilitlemeyi nasıl yapacağımıza bakalım:

UN*X platformunda dosya kilitleme **fcntl** çağrıları kullanılarak yapılır. **fcntl** ile ilgili ayrıntılı dökümanı man sayfalarında bulabilirsiniz. Biz sadece bir örnek vererek dosya kilitlemenin nasıl yapıldığını göstereceğiz. **fcntl** kullanarak dosya kilitleme, **flock** UN*X çağrısı kullanılarak yapılır. Bu çağrıda bir dosya paylaştırılmış ve tam kilitleme ile kilitlenebilir. Paylaştırılmış kilitlemede başka bir kullanıcı aynı dosyayı sadece salt okunabilir açarken, özel kilitleme yönteminde dosya bir başka süreç ya da kişi tarafından hiçbir şekilde açılamaz. UN*X'de dosyanın belirli bir bölümünü de kilitleyebilirsiniz. Dosya kilitleme, dosya tanımlayıcısı üzerine yapılır. Bir dosya açıldığında, o dosyanın küçük bir tamsayı olan dosya tanımlayıcısı (file descriptor) olur. Bu tanımlayıcı üzerine dosya kilidi koyabilir, işiniz bittiğinde de kaldırabilirsiniz.

```
>>> import fcntl
>>> F=open("myfile.txt","w")
>>> fcntl.flock(F.fileno(), fcntl.LOCK_EX)
>>> F.close()
```

Bu programda **open** ile bir dosya açılıyor ve bunun nesnesi **F** değişkenine atanıyor. **fcntl.flock()** dosyaya kilit koyacaktır, ancak hangi dosya tanımlayıcısına ve ne şekilde kilit konulacağı belirtilmelidir. Birinci argüman olarak **F** nesnesinden açılan dosyanın tanımlayıcısını **fileno()** ile öğreniyoruz, ikinci argüman olarak ise **fcntl**'den aldığımız özel kilidi (**LOCK_EX**) koyuyoruz. Dosya kapatıldığında kilit de otomatik olarak kaldırılmış oluyor.

Şimdi de **flcok** kullanarak bir sayacı nasıl yapabileceğimize bakalım:

program11-02.py

```
#!usr/bin/python

import fcntl, os

sayacDosyasi="sayac.txt"
```

```
if not os.path.exists(sayacDosyasi):
        D=open(sayacDosyasi,'w')
        D.write('0')
        D.close()


D=open(sayacDosyasi,"r+")
fcntl.flock(D.fileno(), fcntl.LOCK_EX)
sayac=int(D.readline())+1
D.seek(0)
D.write(str(sayac))
D.close()
```

Yukardaki program üstüne biraz açıklama yapalım: İlk olarak **sayac.txt** dosyasının olup olmadığını kontrol ediyoruz. Eğer yok ise bu dosyayı açıp sayacın ilk değeri olan sıfırı dosyaya yazıyoruz. Daha sonra bu dosyayı hem okunabilir hem de yazılabilir şekilde tekrar açıyoruz. Bir dosyayı bu şekilde açmak için "r+" modu kullanılmalıdır. Dosya açıldıktan sonra onun tanımlayıcısı üzerine bir özel kilit konuluyor. Diğer kilitleme modlarını **dir(fcntl)** ile görebilirsiniz. Sayacın o andaki değeri okunup, üzerine bir ilave edilerek tekrar dosyaya yazılıyor. Dosyaya yazmadan önce tekrar dosyanın ilk karakterine gelmek gerekli. Bu da **seek()** fonksiyonu ile yapılıyor. Eğer bir dosyaya bir karakter yazar veya okursanız, dosya işaretçisi bir sonraki karaktere kayacaktır. Bunun için bu uygulamada dosyanın ilk karakteri olan sıfırıncı karaktere tekrar geri dönmek şarttır.

msvcrt modülü Python içerisinden Microsoft Visual C/C++ Runtime Library fonksiyonlarına ulaşmak için kullanılır. Bu fonksiyonları kullanarak bir dosyayı kilitleyerek, yukarıdaki sayaç programını şu şekilde yazabiliriz:

program11-03.py

```
#!c:/python22/python.exe

import msvcrt, os

sayacDosyasi="c:\\sayac.txt"
```

```
if not os.path.exists(sayacDosyasi):
        D=open(sayacDosyasi,'w')
        D.write('0')
        D.close()


D=open(sayacDosyasi,"r+")
msvcrt.locking(D.fileno(),
               msvcrt.LK_LOCK,
               os.path.getsize(sayacDosyasi))
sayac=int(D.readline())+1
D.seek(0)
D.write(str(sayac))
D.close()
```

Bu programda yapılan işler UN*X'tekine çok benzerdir. Tek farkı **msvcrt**'de **locking**() fonksiyonunun, dosyanın belirli bir bölümü üzerine kilit koymasıdır. Aslında bu UN*X'dede var, ancak genelde dosyanın tamamı üzerine kilit konulduğundan yukarıda anlatmadık. Diğer kilitleme modlarını **dir(msvcrt)** ile görebilirsiniz.

Eğer bu sayaç programını Windows/UN*X platformunda çalışacak şekilde yazsaydık şöyle bir kod gerekliydi:

program11-04.py

```
#!usr/bin/python
import  os

sayacDosyasi="/sayac.txt"

if not os.path.exists(sayacDosyasi):
           D=open(sayacDosyasi,'w')
           D.write('0')
           D.close()
```

Python

```
D=open(sayacDosyasi,"r+")

if os.name=='nt':
    import msvcrt
    msvcrt.locking(D.fileno(),
                   msvcrt.LK_LOCK,
                   os.path.getsize(sayacDosyasi))
else:
    import fcntl
    fcntl.flock(D.fileno(), fcntl.LOCK_EX)

sayac=int(D.readline())+1
D.seek(0)
D.write(str(sayac))
D.close()
```

İnternet Uygulamaları 12

Bu bölümde günümüz dünyasında önemli yer tutan çeşitli internet uygulamalarından bahsedeceğiz. Düşük seviyeli TCP/IP programlamayı anlatmanın yersiz olduğunu düşünüyoruz. Hemen her internet uygulaması için hazır modül ve bunların içerisinde istemediğiniz kadar fonksiyon bulunmaktadır. Basit bir web sunucu yapmaktan, ftp sitelerine erişip dosya indirmeye, pop3 sunucularından mektupları almaya varan birçok uygulamanın Python'da ne kadar kolay yapıldığını göreceksiniz. Yine bu bölümü referans amacı ile kullanmanızı önerirken, yüzeysel olarak da olsa okumanızı tavsiye ederiz.

urllib

urllib modülü http, ftp ve gopher sunucularından verileri alan bir istemcidir. Vereceğiniz adrese göre (url) hangi sunucuya hangi protokol ile erişileceğine otomatik olarak karar verilir. Şimdi bir web sitesinden nasıl veri alınabileceğine bakalım:

```
>>> import urllib
>>> fp=urllib.urlopen("http://localhost")
```

Burada yerel makinede çalışan web sunucusuna bir bağlantı açtık. Henüz hiçbir veri almadık, fakat sunucunun gönderdiği başlıklar alınmış oldu. Başlıklar **fp** nesnesine ait **headers** sözlüğünde duruyor. Bunlara bakalım:

```
>>> for k in fp.headers.keys():
            print k, fp.headers[k]

last-modified Sat, 05 Jan 2002 13:54:10 GMT
content-length 117
```

```
etag "06d2b73f095c11:552"
content-location http://localhost/index.html
date Sat, 05 Jan 2002 15:38:27 GMT
accept-ranges bytes
content-type text/html
server Microsoft-IIS/4.0
```

Şimdi sunucudan okuma işlemi yapalım. Okuma işlemi tıpkı bir dosyadan okurmuş gibi yapılabilir. **read()** veya **readline()**'nı kullanabilirsiniz:

```
>>> data=fp.read()
>>> print data
<html>

<body><head><title>Deneme sayfası</title></head><h1> Kişisel Web
sunucusundaki deneme sayfası</h1>

</body>
```

ftplib

Bir ftp sitesine bağlanıp veri transferi yapacaksanız urllib yerine ftplib kullanmanızı tavsiye ederiz. Bu yüzden yukarda ftp sitelerine olan bağlantıyı anlatmadık. Ftp sitesinde anonim bağlantı yapılabileceği gibi kişisel bağlantı da yapılabilir. Bu amaçla kullanıcı adı ve parolası sunucuya gönderilmelidir. Aşağıda bir ftp sunucusuna nasıl bağlanacağımızı göreceğiz.

192.168.1.1 adresinde bulunan ftp sitesine **mbaser** kullanıcısı olarak girelim. Bu kullanıcının, adı geçen ftp sunucusundaki parolası "yumurcak"tır.

```
>>> import ftplib
>>> ftp=ftplib.FTP("192.168.1.1")
```

Her ftp sunucusu kendisine bağlanan istemciye sunucunun durumuyla birlikte **welcome.msg** dosyasını gönderir. Şimdi buna bakalım:

```
>>> print ftp.welcome
220-=(<*>)=-.:. (( Welcome to PureFTPd 1.0.8 )) .:.-=(<*>)=-
```

```
220-You are user number 4 of 50 allowed
220-Local time is now 20:13 and the load is 0.00. Server port:
21.
220-This is a private system - No anonymous login
220 You will be disconnected after 15 minutes of inactivity.
```

Artık sunucuya kullanıcı adı ve parolasını gönderebiliriz:

```
>>> print ftp.login("mbaser", "yumurcak")
230-User mbaser has group access to:  525
230-OK. Current directory is /
230 0 Kbytes used (0%) - authorized: 10240 Kb
```

Şimdi sunucuda hangi dosya ve klasörlerimizin olduğuna bakalım:

```
>>> print ftp.dir()
drwx------    3 524      525           288 Dec 23 02:30 Desktop
drwx------    3 524      525            72 Dec 14 23:59 Mail
drwxrwxr-x    3 524      525           352 Dec 23 02:03 OpenOf-
fice
-rwxr-xr-x    1 0        0             360 Dec 17 13:08 create-
key.py
-r-xr-xr-x    1 0        0          172096 Oct 14  1998 fips.exe
-rwxr-xr-x    1 0        0           33724 Dec 28 23:01 ftp.so
None
```

fips.exe dosyasını sunucudan alalım. Bu dosya binary olduğundan bu moddaki veri transfer biçimini seçmeliyiz. Önce sunucudan alınan verilerin yazılacağı, binary modda yazılabilir bir dosya nesnesi oluşturmamız gerekir:

```
>>> gelen=open("c:\\fips.exe","wb")
>>> print ftp.retrbinary("RETR fips.exe",gelen.write)
226-File successfully transferred
226 0.009 seconds (measured here), 18.67 Mbytes per second
>>> gelen.close()
```

Şimdi de sunucuya bulunduğumuz klasördeki LPRng-3.6.22-5.i386.rpm dosyasını yükleyelim:

```
>>> dosya="LPRng-3.6.22-5.i386.rpm"
>>> print ftp.storbinary("STOR "+dosya, open(dosya, "rb"), 1024)
26-669.1 Mbytes free disk space
226-1532 Kbytes used (14%) - authorized: 10240 Kb
226-File successfully transferred
226 0.129 seconds (measured here), 11.57 Mbytes per second
```

Son olarak sunucu ile aktif bağlantımızı kesmek için çıkış yapalım:

```
>>> print ftp.quit()
221-Goodbye. You uploaded 1533 and downloaded 169 kbytes.
221 Logout - CPU time spent: 0.010 seconds.
```

poplib

pop3 sunucularına ulaşıp mektupları almak için **poplib** modülü kullanılır. Kullanımı oldukça kolay olan bu modül, bir pop3 istemcisinde bulunabilecek her türlü özelliği kapsıyor. Burada pop3 sunucusuna bağlanıp mektupları nasıl okuyacağımızı göreceğiz.

Bir pop3 sunucusuna ulaşmak için, onun adresini, orada bir kullanıcı adı ve parolasını bilmek gerekir. **mail.easilix.com** adresindeki **mbaser** kullanıcısının pop3 parolası **yumurcak**'tır. Şimdi hemen bağlantı kuralım:

```
>>> import poplib
>>> pop3.user("mbaser")
'+OK '
>>> pop3.pass_("yumurcak")
'+OK '
```

Eğer sunucu sanal e-posta alanları içeriyorsa, kullanıcı adı yerine tam e-posta adresi yazmak gerekebilir. Örneğin **mbaser@easilix.com** gibi. Şimdi sunucuda bulunan mektup listesini alalım:

```
>>> mektupListesi=pop3.list()
>>> print mektupListesi
('+OK ', ['1 663', '2 518'], 14)
```

Sunucuda iki adet mektubun bulunduğunu, birincisinin boyunun 663 byte, ikincisinin ise 518 byte olduğunu görüyoruz. Bu bilgiler gelen verinin ikinci elemanında bir liste halindedir. Liste elemanları birer cümledir ve bu cümleler 'id boyut' şeklindedir. Şimdi ilk mektubu sunucudan okuyalım:

```
>>> resp, metin, oct=pop3.retr('1')
```

metin değişkeninde birinci mektubumuz ham olarak durmaktadır. Bu mektubu parçalarına ayırmak için, **rfc822** modülüne ihtiyacımız vardır. Bu modül, bir mektup dosyasını okuyarak, onu başlıklarına ve gövdesine ayırarak sonucu bir sözlük olarak verir. **rfc822**'nin bir dosyadan mektubu okuduğunu söyledik, ancak bizim mektubuz şu anda bir cümle değişkende duruyor. Ya bu mektubu geçici bir dosyaya yazıp işimiz bittikten sonra sileceğiz ya da hafızada dosya tipi bir nesne oluşturacağız. İkinci yolu kullanmak sistem çağrılarını azaltır ve tercih edilmelidir. Python'da hafızada bir dosya nesnesi oluşturmak için **StringIO** modülü kullanılır.

Şimdi tüm bu açıklamalar ışığında mektubumuzu bir hafıza dosyasına yazalım, sonra da **rfc822** ile bu hafıza dosyasını okuyup parçalarına ayrıştıralım:

```
>>> import StringIO
>>> hafizaDosyasi=StringIO.StringIO("\n".join(metin))
>>> mektup=rfc822.Message(hafizaDosyasi)
```

Dilerseniz mektup sözlüğünün anahtarlarına bir bakalım:

```
>>> mektup.keys()
['subject', 'content-type', 'message-id', 'date', 'mime-version', 'to',
'content-transfer-encoding', 'return-path', 'from', 'delivered-to',
'received']
```

Şimdi bu mektubun başlıklarını yazdıralım:

```
>>> for k in m.keys():
...     print k, "=", m[k]
...
subject = Sevgili Babacigim
content-type = text/plain; format=flowed; charset="iso-8859-1"
message-id = <20020105195157.3531.qmail@mbaser.easilix.com>
date = Sat, 05 Jan 2002 19:51:57 GMT
mime-version = 1.0
to = mbaser@easilix.com
content-transfer-encoding = 8bit
return-path = <fatih@easilix.com>
from = "Fatih BASER" <fatih@easilix.com>
delivered-to = mbaser@easilix.com
received = (qmail 3532 invoked by uid 88); 5 Jan 2002 19:51:57 -
0000
```

Son olarak, mektubun gövdesi:

```
>>> print mektup.fp.read()

Merhaba babacığım,

Uzun zamandır yazışamıyoruz. Bizim derslerimiz çok yoğun. Nasıl-
sınız?

Biz kardeşimle beraber çok iyiyiz, sınavlarımıza girip çıkıyoruz.
Notlarımız da çok iyi. Tatili dört gözle bekliyoruz.

Sizleri çok seven oğlunuz Fatih
```

poplib modülü bu kadar değil. İstediğiniz bir mektubu sunucudan silebilirsiniz. Bunun için **dele()** fonksiyonu kullanabilirsiniz. Şimdi okuduğumuz birinci mektubu sunucudan silmek üzere onu işaretleyelim:

```
>>> pop3.dele('1')
'+OK
```

Silme işaretlerini kaldırmak istiyorsanız **rset()** fonksiyonunu kullanabilirsiniz. Gerçek silme işlemi sunucu ile bağlantı istemcinin arzusuyla koparıldığında gerçekleşir.

Eğer sunucu ile işiniz uzun sürecekse, en iyisi bağlantıyı canlı tutmak için **noop()** fonksiyonunu kullanmaktır:

```
>>> pop3.noop()
'+OK '
```

Hem bağlantıyı koparmak hem de işaretlenmiş mektubun silinmesini gerçekleştirmek üzere, sunucu ile bağımızı sağlıklı bir şekilde keselim:

```
>>> pop3.quit()
'+OK '
```

Pop3 sunucusundan sadece mektup başlığını almak için top(id,0) fonksiyonun kullanabilirsiniz. Böylelikle, mektup listesini oluşturmak için tüm mektupları almanıza gerek kalmaz.



IMAP sunucularına bağlanmak için **imaplib** modülünü kullanmanız gerekir. Kullanımı hemen hemen poplib'e benzer. IMAP sunucuları yaygın olmadığı için bu kitapta anlatılmayacaktır.

smtplib

pop3 sunucusundan mektuplarınızı okuduysanız ve cevap göndermek istiyorsanız, **smtplib** modülünü kullanmanız gerekecek. Şimdi bu modülü nasıl kullanacağımızı basit bir örnekle açıklayalım:

```
>>> import smtplib
>>> smtp=smtplib.SMTP("mail.easilix.com")
>>> smtp.sendmail(
... "From: mbaser@easilix.com",
... ["To: fatih@easilix.com", "To: melike@kolaymail.com"],
```

```
... """From: mbaser@easilix.com
... To: fatih@easilix.com
... Subject: Sevgili oğlum
...
... Çok Sevdiğim Oğlum ve Kızım,
... Mektubunuzda hem sağlığınızın hem de derslerinizin iyi olduğunu
duyduk ve sevindik.
... Ben ve annen iyiyiz. Tatilin gelmesini ve sizlere kavuşmayı
    dört gözle bekliyoruz.
...
... Sizleri çok seven babanız...
... """),
([],)
>>> smtp.quit()
```

SimpleHTTPServer

Python'da web sunucuları oluşturmak için birçok üst düzey modül bulunur. Bunlardan en kolay olanı basit web sunucusu oluşturan **SimpleHTTPServer** modülüdür. Bu modül sayesinde, bir web sunucusu oluşturabilir ve sayfaları yayınlayabilirsiniz. İstemcinin istediği sayfalar, web sunucunun başlatıldığını klasöre göreceli olarak algılanır. Şimdi bir örnek yapalım:

```
>>> import SimpleHTTPServer
>>> import SocketServer
>>> PORT=8080
>>> Handler=SimpleHTTPServer.SimpleHTTPRequestHandler
>>> httpd=SocketServer.TCPServer(("", PORT), Handler)
>>> print PORT, "portunda Python web sunucusu"
8080 portunda Python web sunucusu
>>> httpd.serve_forever()
127.0.0.1 - - [05/Jan/2002 22:26:52] "GET / HTTP/1.1" 200 -
```

Eğer makinenizde bir web sunucusu çalışmıyorsa port'u ön tanımlı değeri olan 80 yapabilirsiniz. Böylece tarayıcıdan port numarası girmezsiniz. UN*X kullanıyorsanız ve program root tarafından çalıştırılmayacaksa 1024'ün altındaki bir portu kullanamazsınız. Eğer örnekte olduğu gibi 8080

portunu kullanacaksanız bu portu başka bir sunucunun dinlemediğinden emin olun.

Basit bir web sayfası yaparak, Python yorumlayıcısını çalıştırdığınız klasöre **index.html** olarak kaydedin. Oluşturduğumuz web sunucusuna bir istek gönderelim. Şimdi Explorer veya Netscape'i çalıştırın ve **http://localhost:8080** adresini girin. Her şey normalse **Şekil-12.1**'deki gibi bir sonuca ulaşmanız gerekiyor.



Şekil-12.1: Python ile bir web sunucusu yapmak çok kolaydır.

webbrowser

Artık herşey web tabanlı. Bu yüzden bir programdan yapması beklenen en doğal şey istenilen bir dökümanı bir web tarayıcısı açtırarak görüntületmesidir. Bu amaçla Python'da **webbrowser** modülü bulunmaktadır. Bu modül kullanıcının kullandığı sistemin ön tanımlı web tarayıcısını çağıracaktır. Windows tabanlı makinelerde bu genellikle Explorer'dir. Basit bir örnek vererek konumuzu burada sonlandıralım:

```
>>> import webbrowser
>>> webbrowser.open("http://www.easilix.com")
```

Çok Kullanılan Standart Modüller

13

Bu bölümde Python programlarında sıkça kullanılan bazı modülleri örnekleriyle birlikte inceleyeceğiz. Vereceğimiz bilgiler hemen hemen her programcının vazgeçemeyeceği hazır kütüphane ve fonksiyonları kapsıyor. Bu bakımdan her ne kadar referans amacı ile kullanılabilirse de dikkatlice okunup anlaşılmasında fayda olacaktır.

tempfile

Birçok programcı verileri geçici bir dosyada saklayıp, dosya ile işi bitince onu siler. Geçici dosyanın nerede ve hangi isimle oluşturulacağı, aynı isimde başka bir dosyanın var olup olmadığını kendisi kontrol eder. Oysaki Python'da bunların hepsi sizin için otomatik olarak yapılır, siz sadece geçici bir dosya isteyin, o kadar. Bu işlemi **tempfile** modülü ile yapabilirsiniz:

```
>>> import os
>>> import tempfile
>>> geciciDosya=tempfile.mktemp()
>>> print geciciDosya
C:\WINDOWS\TEMP\~-1774511-1
>>> D=open(geciciDosya,'w')
>>> D.write("Geçici dosya ile işiniz bitince siliniz.")
>>> D.close()
>>> os.remove(geciciDosya)
```

random

Rastgele sayılar bir programın vazgeçilmez unsurlarıdır. Pythonda rastgele sayıları oluşturmak için **random** modülüne ihtiyacınız olacak. Bu modül sadece sıradan sayılar üretmekle kalmayıp, bir cümle içerisinden sıradan bir

harf veya bir liste içerisinden sıradan bir eleman da seçebilir. Öne sıradan sayıları nasıl elde edeceğimize bakalım:

```
>>> import random
>>> random.random()
0.40627678507039011
>>> random.random()
0.36696193774073826
```

Bazen bir cümle içerisinden rastgele bir harfin seçilmesini isteyebilirsiniz:

```
>>> random.choice('mbaser')
's'
>>> random.choice('mbaser')
'm'
```

Benzer şekilde liste elemanlarından birisini almak isterseniz:

```
>>> L=["beyaz", "mavi", "mor", "siyah"]
>>> random.choice(L)
'beyaz'
>>> random.choice(L)
'mor'
>>> random.choice(L)
'siyah'
```

md5, base64

md5 mesaj imzaları oluşturmak için kullanılır. Bazen kullanıcı parolalarını oluşturmak için de kullanılabilir. **md5** en fazla 128 karakter uzunluktaki kelimeleri imzalayabilir. Bu modül tüm platformlarda vardır. Şimdi basit bir şekilde nasıl kullanılabileceğine bakalım:

```
>>> import md5
>>> hash=md5.new()
>>> hask.update("imzalanacak metin")
>>> hash.hexdigest()
'f9f1dadb4939ca357aa8f48e24014704'
```

Yukarıda hexadecimal olarak yazdırdık. Eğer imzanın ikili (binary) şeklini istiyorsanız **hash.digest()** fonksiyonu ile alabilirsiniz.

md5 ile imzalanan metinler genellikle base64 ile düz metin haline getirilirler. **base64**'ü kullanmak oldukça basittir:

```
>>> import base64
>>> print base64.encodestring(hash.digest())
+fHa20k5yjV6qPSOJAFHBA==
```

Şimdi bir metni **base64** ile şifreleyip geri açalım:

```
>>> B=base64.encodestring("ikimiz bir fidanız")
>>> print B
aWtpbWl6IGJpciBmaWRhbv16

>>> print base64.decodestring(B)
ikimiz bir fidanız
```

pickle

Python'da nesneleri disk üzerinde saklayabilmek için öncelikle onu serileştirmeniz gerekir. Bu işlem pickle modülü ile yapılır. Herhangi bir nesneyi **pickle** ile serileştirebilirsiniz. Bir sözlük üzerinde örnekleyelim:

```
>>> import pickle
>>>sozluk={"mbaser":"mbaser@kolaymail.com", "akgun":"mustafa@ge-
lecek.com.tr"}
>>> seri=pickle.dumps(sozluk)
>>> print seri
(dp0
S'akgun'
p1
S'mustafa@gelecek.com.tr'
p2
sS'mbaser'
p3
```

```
S'mbaser@kolaymail.com'
p4
s.
```

Seri değişkenine atadığımız serileştirilmiş sozluk nesnesini bir dosyaya yazıp tekrar çağırabiliriz.

```
>>> D=open("myfile.txt", "w")
>>> D.write(seri)
>>> D.close()
```

Bir süre sonra bu sözlüğe ihtiyacımız olduğunda:

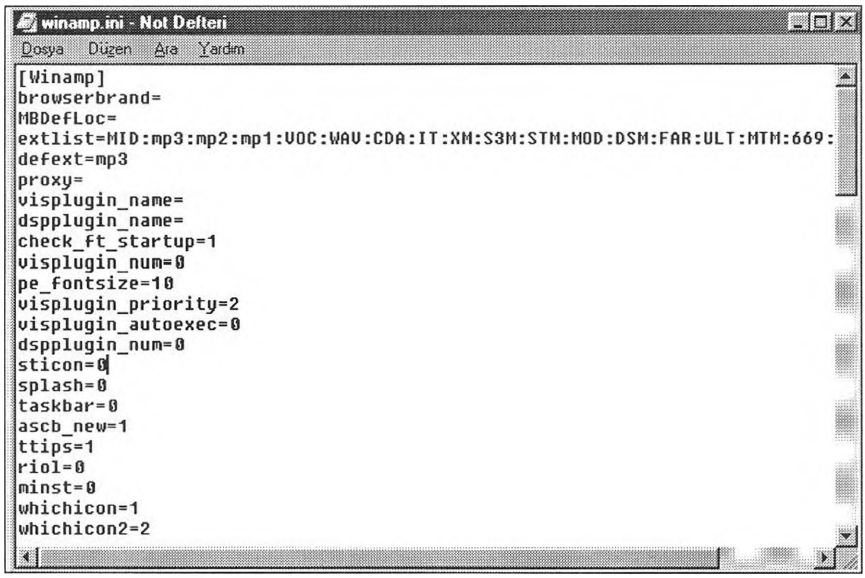
```
>>> s=pickle.loads(open("myfile.txt").read())
>>> s["mbaser"]
'mbaser@kolaymail.com'
>>> print s
{'akgun': 'mustafa@gelecek.com.tr', 'mbaser': 'mbaser@kolayma-
il.com'}
```

ConfigParser

Genel amaçlı yazılmış programlar başladıklarında genellikle bir yapılandırma dosyası okuyarak program akışını ona göre belirler. Windows üzerindeki yapılandırma dosyaları genellikle program.ini isimleri ile kaydedilirler ve yapıları şu şekildedir:

```
[bolum]
"secenek1" = "deger1"
"secenek2" = "deger2"
```

Hemen hemen tüm Windows'larda eksik olmayan Winamp programının yapılandırma dosyası **winamp.ini**'nin bir kısmını **Şekil-13.1**'de görüyorsunuz:



Şekil-13.1: winamp.ini dosyası

Bu şekildeki yapılandırma dosyalarını okuyup yeniden yazmak çok zahmetlidir. Python'da ise bu işlemi **ConfigParser** modülü ile rahatça yapabilirsiniz. Şimdi **winamp.ini** dosyasını açalım ve bölümler ile değerlerini ekrana yazdıralım:

```
>>> import ConfigParser
>>> config=ConfigParser.ConfigParser()
>>> config.read("c:\\program files\\winamp\winamp.ini")
>>> config.sections()
['In_asfs', 'Winamp']

>>> for b in config.sections():
            print
            print "Şimdi %s bölümü yazılıyor\n" % b
            for s in config.options(b):
              print s," -->", config.get(b,s)
```

```
Şimdi In_asfs bölümü yazılıyor

songspath  --> C:\PROGRAM FILES\WINAMP\

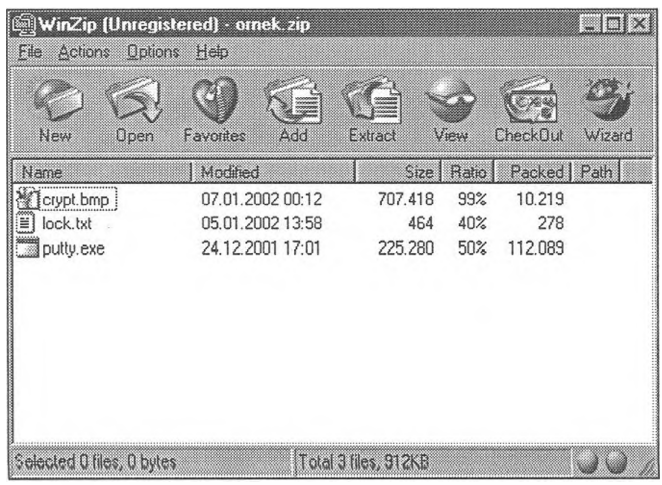
Şimdi Winamp bölümü yazılıyor

shuffle  --> 0
visplugin_name  -->
dspplugin_name  -->
preamp  --> 31
keeponscreen  --> 1
last_shortdesc  -->
.
.
.
```

zipfile

Zipfile modülü Winzip veya benzeri bir sıkıştırma aracı ile sıkıştırılmış zip dosyalarını sorgulamak, okumak ve yazmak için kullanılır.

Bir zip'in içerisindeki dosyaların listesini almak için, **namelist()** fonksiyonu kullanılır. **Şekil-13.2**'de **WinZip** ile görünen ornek.zip dosyasının içerisindeki dosyalara Python ile bakalım:



Şekil-13.2: ornek.zip dosyasının içeriğini WinZip ile görüntülenmesi

```
>>> import zipfile
>>> zip=zipfile.ZipFile("c:\\windows\\desktop\ornek.zip")
>>> print zip.namelist()
['crypt.bmp', 'lock.txt', 'putty.exe']
```

Zip içerisindeki dosyaların oluşturma tarihleri, büyüklükleri ve sıkıştırılış büyüklükleri ile ilgili bilgi de alabilirsiniz. Bunun için **infolist()** nesnesi kullanılır:

```
>>> for b in zip.infolist():
            print b.filename, b.date_time, b.file_size, b.comp-
ress_size


crypt.bmp (2002, 1, 7, 0, 12, 40) 707418 10219
lock.txt (2002, 1, 5, 13, 58, 10) 464 278
putty.exe (2001, 12, 24, 17, 1, 34) 225280 112089
```

Zip'in içerisindeki dosyalardan birisini almak oldukça kolay. **Ornek.zip** dosyasındaki **putty.exe** dosyasını alıp **c:\putty.exe** olarak kaydedelim ve zip dosyasını kapatalım. Yazmak için açacağımız dosyanın ikili (binary) olmasına dikkat edin:

```
>>> D=open("c:\\putty.exe","wb")
>>> D.write(zip.read("putty.exe"))
>>> D.close()
>>> zip.close()
```

Zip arşivine bir dosya eklemek için onu ekleme (append'a) modunda açmanız gerekecektir. Eğer yeni bir arşiv yapacaksanız o zaman yazma (write, w) modunda açabilirsiniz. Şimdi **ornek.zip** arşivine **c:\windows\desktop\urunler.xls** dosyasını ekleyelim:

```
>>> zip=zipfile.ZipFile("c:\\windows\\desktop\ornek.zip","a")
>>>  zip.write("c:\\windows\desktop\urunler.xls",  "urunler.xls",
zipfile.ZIP_DEFLATED)
>>> zip.close()
```

crypt, getpass

UN*X dünyasının vazgeçilmez şifreleme yöntemi crypt tersinemez bir şifrelemedir. Bu fonksiyon **crypt** modülü içerisindedir. Bu modül Windows platformunda yoktur. En fazla 8 karaktere kadar olan kelimeleri şifrelerken sizden iki karakterden oluşan bir de anahtar isteyecektir. Genellikle kullanıcı parolaları bu fonksiyon ile şifrelenir. Tersine dönüş olmayacağına göre kullanıcının parolası nasıl doğrulanacak? Bu sorunun yanıtını birazdan alacaksınız.

Şifreleme yapılırken iki karakterlik anahtar genellikle rastgele seçilir. Bu işlemi, yukarıda gördüğümüz random modülü ile yapabiliriz. Tüm harfler string modülünden çağrılabilir:

```
>>> import string
>>> import crypt
>>>anahtar=random.choice(string.letters)+random.cho-
ice(string.letters)
>>> anahtar
'tG'
>>> sifrelenmisParola=crypt.crypt("yumurcak",anahtar)
>>> sifrelenmisParola
'tGm8vYvw36O62'
```

Yukarıda da görüldüğü gibi şifrelenmiş metnin ilk iki karakteri, anahtardan oluşmaktadır. Bu anahtar sayesinde parola doğrulamaları yapılabiliyor. Şimdi, gölge parola kullanan bir Linux'ta kayıtlı mbaser kullanıcısının girdiği parolanın, doğru olup olmadığını denetleyelim. Önce **/etc/shadow** dosyasını açıp mbaser kullanıcısının şifrelenmiş parolasını bulalım ve diskParola değişkenine atayalım:

```
>>> diskParola=''
>>> for p in open("/etc/shadow").readlines():
     sp=p.split(':')
     if sp[0]=='mbaser':
            diskParola=sp[1]
            break
```

```
>>> diskParola
'ZFORGAlg4ee9A'
```

Kullanıcının gönderdiği parola "yumurcak" olsun. Bu parolayı diskteki parolanın ilk iki karakterine göre şifreleyelim. Eğer yeni elde ettiğimiz şifrelenmiş parola diskteki ile aynı ise, o halde kullanıcının girdiği parola doğrudur:

```
>>> if crypt.crypt("yumurcak", diskParola[:2]) == diskParola:
print "Parolanız doğrulandı."

Parolanız doğrulandı.
```

Şimdi bunların hepsini bir araya getirelim ve bir onaylama modülü yazalım. Bu modül, diğer programlar tarafından kullanıcıların onaylanması amacı ile kullanılabilir:

program13-1.py

```
#!/usr/bin/python2

"""
Bu modül, shadow ve crypt kullanan UN*X işletim sistemlerinde,
kullanıcıların onaylanması için kullanılabilir. Onayla fonksiyonu iki parametre alır. İlki kullanıcı adı ikincisi paroladır. Bir
tüp halinde olan geri dönüşümün ilk elemanı 1 ise kullanıcı
onaylanmıştır. İkinci eleman ise açıklamadan oluşur. Örnek kullanımı:

            >>> import mod_onayla
            >>> sonuc=mod_onayla.onayla("mbaser", "yumurcak")
            >>> print sonuc[0], sonuc[1]
            1 Tebrikler, onaylandınız

"""
import crypt, getpass
```

Python

```
def onayla(kullaniciAdi, parola):

            diskParola=''
            for p in open("/etc/shadow").readlines():
              sp=p.split(':')
              if sp[0]==kullaniciAdi:
                     diskParola=sp[1]
                     break

            if not diskParola:
              return (0, 'Bu kullanıcı kayıtlı değil')
            if crypt.crypt(parola, diskParola[:2]) == diskParola:
              return (1, 'Tebrikler, onaylandınız')

            return (0, 'Üzgünüz, parolanız yanlış')



if __name__ == '__main__':

            kullaniciAdi=raw_input('Kullanıcı adınızı yazınız: ')
            parola=getpass.getpass('%s için parola giriniz: ' %
kullaniciAdi)

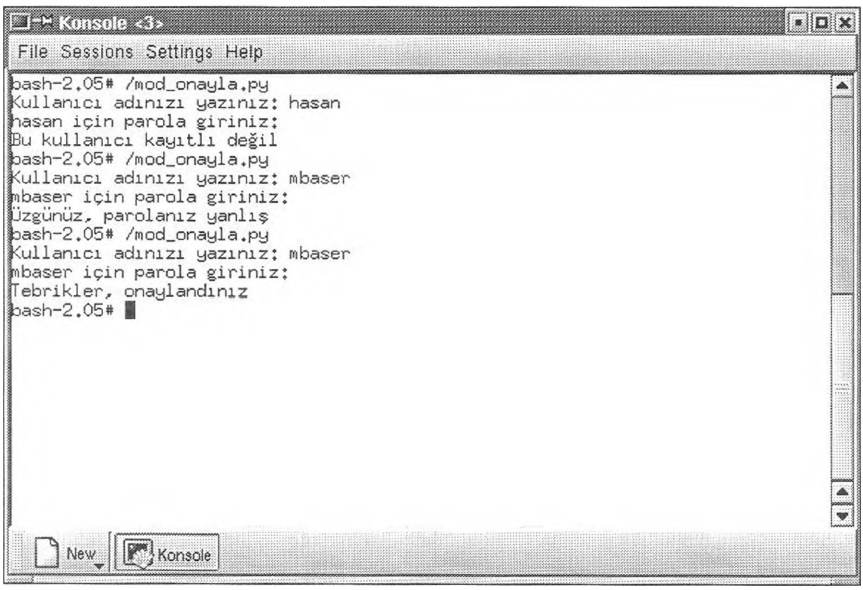
            print onayla(kullaniciAdi, parola)[1]
```

Bu programda **getpass** modülünü de kullanmış olduk. Bu modül, konsoldan parola okumak için kullanılabilir. Kullanıcı parolasını yazarken ekranda hiçbir şey gözükmeyecektir. Bu programı çalıştıracak olursak:

```
bash-2.05# /mod_onayla.py
Kullanıcı adınızı yazınız: hasan
hasan için parola giriniz:
Bu kullanıcı kayıtlı değil
```

```
bash-2.05# /mod_onayla.py
Kullanıcı adınızı yazınız: mbaser
mbaser için parola giriniz:
Üzgünüz, parolanız yanlış
bash-2.05# /mod_onayla.py
Kullanıcı adınızı yazınız: mbaser
mbaser için parola giriniz:
Tebrikler, onaylandınız
```

Bu çıktının ekran görüntüsünü **Şekil-13.3**'de görüyorsunuz.



Şekil-13.3: Onaylama modülünün program olarak çalıştırılması

Veritabanı-1: shelve



Python'un hemen hemen her SQL vertabanı sunucusuyla uyumlu bir şekilde çalışabildiğini söylemiştik. Bununla da kalmayıp ODBC desteği ile her türlü veritabanı sunucusuna rahatça erişebilir. COM uzantısı sayesinde Excel ve Access dosyalarını okuyup sorgulamalar yapabilir ve dosyaların üzerine yazabilir.

Python programcıları çoğu zaman iç tanımlı vertabanı sistemini kullanarak dış bir SQL kaynağına pek başvurmazlar, ancak diğer uygulamalarla bütünleşik çalışmak için bir veritabanı ile çalışmak zorunda kalabilirsiniz. Biz önce Python'un kendi bünyesinde bulunan veri saklama metodunu açıkladıktan sonra, en çok kullanılan sunucu olan MySQL'den bahsedeceğiz.

shelve

Python dbm ve gdbm dosyalarını okuyup yazabilir. Fakat bunlarda anahtar bir cümle ve değerlerde yine bir cümle olabilir. Dbm ile Python nesne taşıyıcısı pickle'ın birleşmesinden meydana gelen **shelve** en yaygın olarak kullanılan veri saklama modülüdür.

Birçok veritabanı programında veriler bir anahtara göre depolanıp tekrar çağırılır ve üzerinde değişiklik yapıldıktan sonra tekrar kaydedilir. Eğer böyle bir sisteme ihtiyacınız var ise, herhangi dış bir kaynağa ihtiyacınız olmadan bütün işlemleri Python içerisinden halledebilirsiniz. Fakat bunun dışındaki uygulamalar için mutlaka sağlam bir veritabanı sunucusuna ihtiyacınız olacak.

shelve anahtar:değer şeklinde çalışan bir nevi kalıcı sözlükler olarak düşünülebilir. Hemen bir **shelve** dosyası açıp bir değer kaydedelim:

```
>>> import shelve
>>> dbf=shelve.open("ornek.db")
>>> dbf["mbaser"]="mbaser@kolaymail.com"
>>> dbf["akgun"]="mustafa@gelecek.com.tr"
>>> dbf.close()
```

Bu programda **ornek.db** veritabanı dosyası oluşturuldu ve içerisine **mbaser** ile **akgun** kullanıcılarının e-posta adresleri yazıldı. Shelve ile bir dosyayı aynı anda hem okunabilir hem de yazılabilir şekilde açabilirsiniz. Böylelikle yaptığınız tüm değişiklikler otomatik olarak diske kaydedilir. Shelve ile açılan bir dosya mutlaka kapatılmalıdır. Her ne kadar kapatmayı unuttuğunuz dosyalar Python'un çalışması sonlanınca otomatik olarak kapatılsa da, programın istem dışı sonlandırılması durumunda veritabanı dosyanız bozulabilir veya son değişiklikler kaydedilmemiş olabilir. Bir dosyayı açıp hemen kapamanızda fayda vardır.

Şimdi yukarıdaki dosyayı tekrar açalım, **gorkem** kullanıcısını ekleyelim ve Mustafa'nın e-posta adresine bakalım:

```
>>> dbf=shelve.open("ornek.db")
>>> dbf["gorkem"]="gorkem@gelecek.com.tr"
>>> dbf["akgun"]
'mustafa@gelecek.com.tr'
>>> dbf.close()
```

Gördüğünüz gibi sanki bir sözlükle oynarcasına disk üzerinde kayıtlı olan verileri okuyor ve eklemeler yapıyoruz. İsterseniz değişiklikler de yapabilirsiniz:

```
>>> dbf=shelve.open("ornek.db")
>>> dbf["mbaser"]="bmustafa@easilix.com"
>>> dbf.close()
```

Yukarıda **mbaser** kullanıcısının e-posta adresinde değişiklik yaptık. Yaptığımız tüm değişiklikler hiçbir çaba sarfetmeden diske yazılmaktadır. Hemen bir uyarıda bulunalım: *Bir shelve dosyası yazmak üzere sadece ve sadece tek bir süreç tarafından açılmalıdır.* Çoklu okumalar güvenli olmasına rağmen

yazmalar da veri kaybı ya da dosyanın bozulma riski vardır. Shelve ile dosyaları açarken herhangi bir açma modu vermezseniz hem okunabilir hem de yazılabilir modda açılır. Şimdi sadece okuma modunda açalım, önce okuyup sonra değişiklik yapmaya çalışalım:

```
>>> dbf=shelve.open("ornek.db","r")
>>> dbf["mbaser"]
'bmustafa@easilix.com'
>>> dbf["mbaser"]="mbaser@easilix.com"
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in ?
    dbf["mbaser"]="mbaser@easilix.com"
  File "C:\PYTHON22\lib\shelve.py", line 77, in __setitem__
    self.dict[key] = f.getvalue()
error: (1, 'Operation not permitted')
```

Sadece okumak üzere açtığınız dosyaları kapatmanıza gerek yoktur ve güvenlidir.

Bir anahtarı ve dolayısıyla onunla ilgili bilgileri silmek de oldukça kolaydır. Bunu Python da bir nesneyi silmek için kullanılan **del** fonksiyonu ile yaparsınız:

```
>>> dbf=shelve.open("ornek.db")
>>> del dbf["gorkem"]
>>> dbf.close()
```

Bir veritabanı dosyasındaki anahtarları **keys** ile, kaç tane kayıt olduğunu da **len()** fonksiyonu ile görebilirsiniz:

```
>>> dbf=shelve.open("ornek.db","r")
>>> dbf.keys()
['akgun', 'mbaser']
>>> len(dbf)
2
```

Kayıt sayısını Python2.2 > için, dbf.__len__() ile de görebilirsiniz.

Eğer yeni bir kayıt girilecekse, daha önce o anahtarın var olup olmadığı mutlaka kontrol edilmelidir. Bunu **has_key()** fonksiyonu ile yapabilirsiniz:

```
>>> dbf=shelve.open("ornek.db","r")
>>> dbf.has_key("gorkem")
0
>>> dbf.has_key("mbaser")
1
```

Shelve Uygulaması

Yukarıda öğrendiğimiz bilgiler ışığında hemen anahtar-değer tipi çalışan bir sistem kurgulayalım. Bir sınıfın öğrencilerini düşünelim. Her öğrenci bir okul numarasına sahiptir ve bütün kütük bu numara ile takip edilir. Aynı isme sahip birkaç öğrenci olsa da numaraları farklı olduğundan, öğrenciler birbirlerinden rahatlıkla ayırt edilebilirler. O halde her öğrenciyi tanımlayan ve sadece bir öğrenciye ait olan tek şey onun numarasıdır. Bu numara sayesinde öğrencinin notları, devamsızlığı ve diğer bilgileri denetlenebilir. Biz basit olsun diye, sadece öğrencilerin metematik, fen ve sosyal bilgiler derslerinin kaydını tutan bir sistem tasarlayalım:

öğrenci no : adı : dersler: notlar

Eğer böyle bir sözlük tasarlayabilirsek, bu sözlüğü diske yazıp istediğimiz zaman rahatlıkla okuyup, hiçbir zahmete katlanmadan değişiklikleri yapıp kaydedebilirsek tüm sorunumuz çözülmüştür. Sözlüğümüzün anahtarı öğrenci no'su olacak. Bu sözlüğün değeri yine bir sözlük olacak. Şimdi bir öğrenci üzerinde bunu yapalım:

```
"392":{
"adı":"Fatih Başer",
"dersler":{
                "matematik":9,
                "fen":10,
                "sosyal":7
        }

}
```

Yukarıdaki bilgiler biraz karışık gibi gelebilir. Kullandıkça ne kadar kolay olduğunu göreceksiniz. Şimdi bu bilgileri bir veritabanına yazalım:

```
>>> dbf=shelve.open("2A.db")
>>> dbf["392"]={"adı":"Fatih Başer", "dersler": {"matematik":9,
"fen":10, "sosyal":7}}
>>> dbf.close()
```

Sağ salim dosyamızı oluşturup verilerimizi sakladık. Artık istediğimiz zaman dosyayı açıp bilgilere ulaşabilir ve değişiklikleri yapabiliriz. Şimdi dosyayı açalım, önce 392 numaralı öğrenicinin adını ve fen dersindeki notunu öğrenelim, sonra da yeni kayıtlar ekleyelim:

```
>>> dbf=shelve.open("2A.db")
>>> print dbf["392"]["adı"], dbf["392"]["dersler"]["matematik"]
>>> print dbf["392"]["adı"], dbf["392"]["dersler"]["fen"]
>>>  dbf["278"]={"adı":"Ufuk Dağcı","dersler": {"matematik":8,
"fen":6, "sosyal":10}}
>>> dbf["291"]={"adı":"Hakan Kara", "dersler": {"matematik":4,
"fen":5, "sosyal":5}}
>>> dbf.close()
```

Bu tür veritabanlarında bir anahtara ait değerde değişiklik yapacağınız zaman önce o değeri bir değişkene atayıp, istediğiniz değişiklikleri yaptıktan sonra da değişkeni anahtara atamanız gereklidir. Aksi halde yaptığınız değişiklikler kaydedilmeyecektir. 392 no'lu öğrencinin, fen notunda bir değişiklik yapalım:

```
>>> dbf=shelve.open("2A.db")
>>> x=dbf["392"]
>>> x["dersler"]["fen"]=10
>>> dbf["392"]=x
>>> dbf.close()
>>> dbf=shelve.open("2A.db",'r')
>>> dbf["392"]["dersler"]
{'fen': 10, 'matematik': 9, 'sosyal': 7}
```

2A sınıfının metematik dersinin listesini çıkartmak istersek:

```
>>> dbf=shelve.open("2A.db",'r')
>>> sinifListesi=dbf.keys()
>>> sinifListesi.sort()
>>> for no in sinifListesi:
      print no, dbf[no]["adı"], dbf[no]["dersler"]["matematik"]


278 Ufuk Dağcı 8
291 Hakan Kara 4
392 Fatih Başer 9
```

Veritabanı-2: MySQL

15

Eğer önceki bölümde verdiğimiz örnekteki gibi bir veritabanı uygulaması sizi tatmin etmiyorsa dış bir SQL sistemi kullanmanız gerekecek. Bu sistem Python ile yazılmış bir SQL sunucusu olabildiği gibi, MsSQL, Oracle, Sybase, PostgreSQL veya MySQL olabilir. Kitabımızda çok yaygın olarak kullanıldığından MySQL ile olan bağlantılardan bahsedeceğiz. Diğer sistemlerde de benzer şekilde kullanılmaktadır.

Peki yukarıda verdiğimiz örnekte sizi tatmin etmeyecek ne olabilir?

- Kayıt sayısı yüz binlerle ifade edilmişse,
- Sorgulamalar tekdüze değil, belirli bir aralıkta veya belirli bir noktada ise: Örneğin matematik notu 9 olanlar veya fen notu 5'den küçük, sosyal notu 6'dan büyük olanların listesi gibi çapraz sorgulamalar yapacaksanız. (Aslında yazacağınız ek fonksiyonlarla benzer şeyleri yukarıdaki program ile de yapabilirsiniz, ancak kullanıcı sayısı yüzbinlere ulaştığında sistem çok yavaşlayacaktır. Hem neden eziyet çekesiniz?)
- Aynı anda binlerce kayıt girme işlemi yapılacaksa,
- Veriler sadece Python'da değil başka programlama dilleri veya programlar tarafından da kullanılacaksa (Aslında yazılacak basit istemci sunucu ile bunun da üstesinden gelebilirsiniz. Neden eziyet çekelim?),
- Veriler dağıtık sistemlerde kullanılacaksa,
- Replication gerekliyse,

Eğer yukarıdaki çekinceleriniz varsa o zaman buyurun SQL dünyasına.

MySQL Kurulumu

Eğer Windows kullanıyorsanız, web programcılığına geçtiğiniz zaman ekstra şeyler istemiyorsanız, PHP ve Perl ile birlikte Apache+MySQL sunucu-

ları ve MySQL yönetim araçlarından oluşan AbriaSoft'un paketlediği **abriamerlin.exe**'yi kurmanızı tavsiye ederiz. Bu paket ihtiyacınız olan hemen her yazılımı sisteminize otomatik bir şekilde kuracaktır. Kurulumu çok kolay olduğundan anlatma gereği duymuyoruz. Kurarken sizden istediği MySQL root parolasını sakın unutmayın.Yazılımı *http://www.abriasoft.com/* adresinden indirebilirsiniz.

MySQL'i kurduktan sonra Python kütüphanesinide kurmanız gerekecek. İndirebileceğiniz adres: *http://sourceforge.net/projects/mysql-python*. Bunun kurulumu da oldukça basittir.

UN*X ya da herhangi bir Linux kullanıyorsanız, büyük bir ihtimalle sisteminizde zaten MySQL kuruludur. Kurulu değilse, elinizdeki CD'lerden veya *http://www.mysql.org* sitesinden platformunuza uygun bir dağıtımı alabilir veya kaynak kodlarını indirerek derleyip kurabilirsiniz. Python bağlantısını da yukarıdaki adresten alabileceğiniz gibi, elinizde CD'ler içerisinde bulabilirsiniz. Red Hat 7.x kullanıcıları ihtiyaç duydukları tüm yazılımları CD'lerde bulabileceklerdir. Kurulum yine birkaç paketi sisteme açmaktan ibaret olduğundan burada anlatmayacağız.

MySQL vertitabanı sisteminin tamamını root şifresi ile değiştirebilirsiniz. Yeni veritabanı açıp silebilir, yeni kullanıcılar açabilir veya üzerinde değişiklikler yapabilirsiniz. Bu yüzden root parolası çok önemlidir ve sizden başkasının bilmemesi gerekir. MySQL Windows üzerine kurulduysa sizden kurulurken bu parolayı istemiş olacaktır. Eğer yeni bir Linux ya da UN*X kurduysanız ve henüz root parolasını değiştirmemişseniz, bunu hemen değiştirmelisiniz.

Aşağıda anlatacağımız MySQL komutları tüm platformlar için geçerlidir.

Sunucuya komut satırından bağlanmak için kullanacağınz program **mysql**'dir. Bu program sayesinde tablolarda değişiklik yapabilir, kullanıcı açabilir veya silebilirsiniz. Veritabanı eklemek ve silmek içinse **mysqladmin** komutunu kullanmanız gerekecek. Windows kullanıcıları eğer AbriaSoft'u kurmuşlarsa bu ve diğer MySQL komutlarını "**C:\Program Files\Abria Merlin\MySQL\bin**" klasöründe bulabilirler.

Bir kişinin MySQL'i kullanabilmesi için öncelikle, sunucuda tanımlı bir kullanıcı adının olması ve bu kullanıcının okuma veya yazma yetkisinin bulunduğu bir veritabanının olması gerekir.

Windows'da eğer MySQL başlangıçta açılmamışsa "Başlat / Programlar / AbriaSoft / Abria SQL (MySQL) / Start MySQL" ile başlatabilirsiniz.

MySQL Komut Satırı ve İlk Bağlantı

Hemen hemen tüm SQL sunucularında komut satırı aynıdır ve birçok komut tüm sunucular için geçerlidir. Eğer önceden herhangi bir sistem ile çalışmışsanız ve MsAccess'i komut satırından kullanmışsanız, bu yapı size çok zor ve karmaşık gelmeyecektir. Diğer okuyucularımız ise ilk kez karşılaşacakları için önce biraz yabancılık çekebilirler.

Komut satırı **mysql>** imleci ile kendisini belirtir. Komut satırında bir sonraki satıra kelimelerin ayrıştığı yerden geçebilirsiniz. Bir satırın (daha doğru bir ifadeyle bir sorgunun) bitimini noktalı virgül (;) ile belirtirsiniz. Satır sonu verilmedikçe sorgunun devam ettiği anlaşılır ve hiçbir işlem yapılmaz. Herhangi bir zamanda "help" yazarak basit bir yardım çağırabilirsiniz.

MySQL sorgularında büyük küçük harf duyarlılığı yoktur, fakat anlaşılırlık kolay olsun diye kimi yazarlar (bazen de biz) SQL terimlerini büyük harflerle yazmayı yeğlerler.

MySQL'e komut satırından **mysql** komutu ile bağlanabilirsiniz. Bu komutun basitçe kullanımı şöyledir:

```
mysql –u <kullanıcıAdı> -p
```

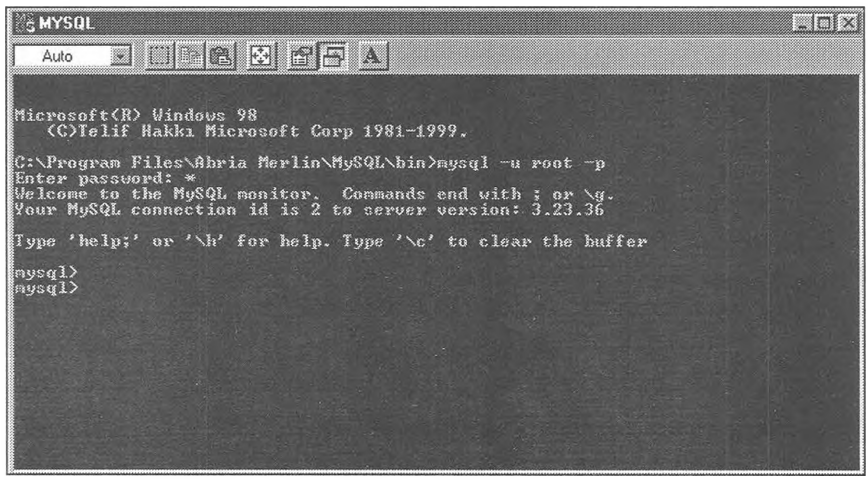
Eğer kullanıcının parolası yapılandırılmamışsa "–p" parametresini kullanmanıza gerek yoktur.
Bağlantı yapacağınız sunucu bulunduğunuz makinede değil de uzaktaki bir sistemde bulunuyorsa, makinenin IP adresini de "–h" parametresi ile belirtmelisiniz:

```
mysql –u <kullanıcıAdı> -h <sunucunun IP adresi> -p
```

Şimdi bulunduğumuz makinede çalışan sunucuya ilk bağlantıyı gerçekleştirelim:

```
C:\Program Files\Abria Merlin\MySQL\bin>mysql -u root -p
Enter password: *
```

Bağlantı sağlıklı bir şekilde kurulmuşsa **Şekil-15.1**'deki gibi bir ekranla karşılaşmanız gerekiyor.



Şekil-15.1: Komut satırından MySQL bağlantısı

Sunucuya bağlantı kurulduktan sonra üzerinde çalışacağınız veritabanını

mysql> USE <veritabanı>;

komutu ile açabilirsiniz.

MySQL'de Veritabanı Açılması ve Silinmesi

Kullanıcı açmadan önce onun kullanacağı veritabanını açmanız gerekiyor. Kitabı sıra ile takip ediyorsanız ve yukarıdaki gibi sunucuya bağlantı kurmuşsanız, veritabanını açmak için

mysql> CREATE <veritabanıİsmi>;

Komutu ile yeni bir vertabanı açabilirsiniz. Şimdi **siniflar** ve **deneme** isimli iki veritabanı açalım:

```
mysql> create database siniflar;
Query OK, 1 row affected (0.00 sec)
mysql> create database deneme;
Query OK, 1 row affected (0.00 sec)
```

Sistemde var olan veritabanlarını görmek için:

```
mysql> show databases;
+-------- +
| Database |
+--------- -+
| deneme   |
| mysql    |
| siniflar |
| test     |
+--------- +
4 rows in set (0.00 sec)
```

Bir veritabanını silmek için:

mysql> DROP DATABASE <veritabanıİsmi>

Yukarıda açtığımız **deneme** veritabanını silelim:

```
mysql> DROP DATABASE deneme;
Query OK, 0 rows affected (0.00 sec)
```

MySQL'de Kullanıcı Açmak

Eğer konuyu sırası ile takip ediyorsanız yukarıdaki gibi bağlantıyı gerçekleştirdiğinizi varsayarak bir kullanıcının nasıl açılabileceğine geçelim. MySQL kullanıcıları çeşitli yetkilere sahip olabilirler. Bu yetkiler veritabanları üzerinde yazma, değişiklik yapma, okuma gibi çeşitli özelliklerde olabilir. Kitabımız MySQL kitabı olmadığından sadece bir veritabanı üzerinde tüm yet-

kilere sahip bir kullanıcının nasıl açılabileceğiniz göstereceğiz. Kullanıcı adı **mbaser**, parolası **yumurcak** olsun. Bu kullanıcı yukarıda açmış olduğumuz **siniflar** veritabanına yerel makineden ulaşarak, üzerinde tam yetkiye sahip olsun:

```
mysql> GRANT ALL PRIVILEGES ON siniflar.* TO mbaser@localhost
IDENTIFIED BY
    -> 'yumurcak' WITH GRANT OPTION;
Query OK, 0 rows affected (0.16 sec)
```

Eğer bu kulllanıcının tüm veritabanlarında tam yetkili olmasını istiyorsanız "**siniflar.***" yerine "***.***" yazabilirsiniz. Başka bir makineden erişmesini istiyorsanız "**mbaser@localhost**" yerine "**mbaser@<IPAdresi>**" şeklinde bir yazılım yapabilirsiniz. Eğer tüm dünyadan ulaşmasını istiyorsanız o zaman "**mbaser@'%'**" yapabilirsiniz.

Tablo İşlemleri

Verilerin asıl saklandığı yer veritabanındaki tablolardır. Bu tabloları aynen Excel tabloları gibi düşünebilirsiniz. Tek farkı MySQL'de her sütunda saklanacak veritipinin önceden belirli olmasıdır. Veri tipleri çok fazladır, fakat aşağıda anlatacaklarımız birçok programcı için yeterli olacaktır. Şimdi MySQL'deki veri tiplerine bakalım ([] içerisindeki seçenekler isteğe bağlıdır):

INT [(<buyutu>)]: Tamsayı veri tipini belirtir. Eğer saklanacak tamsayıların uzunluğu hakkında bir bilginiz var ise, performansı artırmak için boyutunu vermekte fayda vardır.

BIGINT [(<boyutu>)]: Büyük tamsayılar.

REAL [(<boyutu>)]: Reel sayılar. Boyut belirtmek performansı artırır.

CHAR: Sadece bir karakter saklamak için kulanabilirsiniz.

VARCHAR (<boyutu>): Cümle değerleri saklayabileceğiniz veri tipidir. Tabloda saklayacağınız satır sayısı fazla olacaksa (örneğin yüzbinden fazla) mutlak surette boyut belirtiniz. Yoksa performansda önemli düşüşler mey-

dana gelir. Bu veri tipi 128 karakterden daha uzun cümleleri taşımaz. Yerine TEXT kullanmalısınız.

TIME: Tarih bilgisini saklar.

TEXT: Metinleri saklamak için kullanabilirsiniz.

Shelve konusunda verdiğimiz sınıf örneğini burada da vereceğiz. Öncelikle sütunlarımızı ve veri tiplerini belirleyelim.

No: Tamsayı değerlerden oluşan öğrenci numarası saklanacaktır. Bir okul için numaralar genelde en fazla dört hane olur. O halde: **INT(4)**

Adisoyadi: Öğrencinin isim ve soyisminini bu sütunda saklayacağız. Yapılan araştırmalarda isim ve soyismin 50 karakterden daha az olduğu saptanmıştır. Buna göre: **VARCHAR(50)**

Matematik, Fen, Sosyal: Bu sütunlara öğrencinin derslerden aldığı notları yazacağız. Notlar tamsayı değerlerden oluşmakta ve en fazla iki hane olabilmektedir: **INT(2)**

Tüm bunların yanısıra henüz bahsetmediğimiz bir de **primary key** ifadesi var. Shelve konusunda, öğrencilerin numaraları ile takip edildiklerini, bir öğrenciye ait olan numaranın başka birine ait olamayacağını söylemiştik ve sorgulamalarımız hep öğrenci numarası üzerine yapılmıştı. O halde burada "no" sütunu eşsiz olmalı ve performans açısından indexleme buraya göre yapılmalıdır. Tüm bu bahsettiklerimizi **primary key** ifadesi yapmaktadır.

MySQL'de tablo oluşturmak için:

mysql> CREATE TABLE <tabloİsmi>
(<sütunAdı> <veritipi>,<sütunAdı> <veritipiA> ...);

şeklinde bir yazılım kullanılır. Bu açıklamalar ışığında "2A" tablosunu oluşturalım:

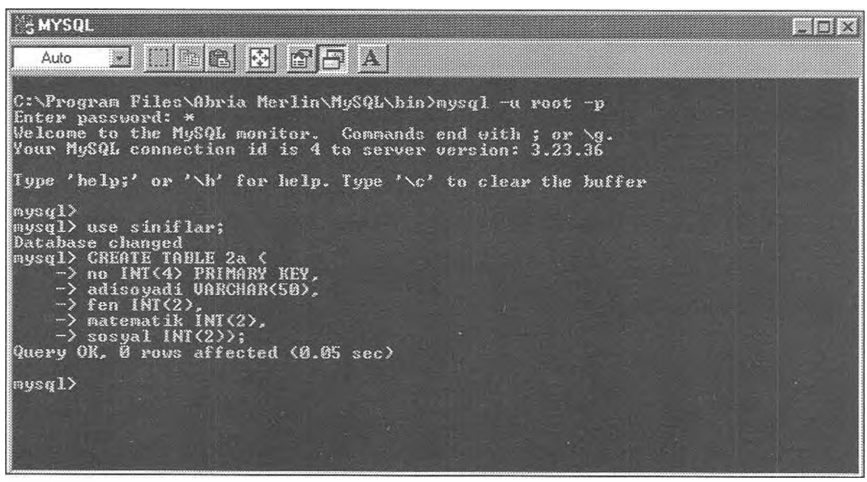
Python

```
C:\Program Files\Abria Merlin\MySQL\bin>mysql -u mbaser -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 22 to server version: 3.23.36

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>
mysql> use siniflar;
Database changed
mysql> CREATE TABLE 2a (
    -> no INT(4) PRIMARY KEY,
    -> adisoyadi VARCHAR(50),
    -> fen INT(2),
    -> matematik INT(2),
    -> sosyal INT(2));
Query OK, 0 rows affected (0.05 sec)
```

Yukarıda yaptığımız işlemleri **Şekil-15.2**'de görüyorsunuz.



Şekil-15.2: MySQL'de ilk tablonun oluşturulması

Bulunduğumuz veritabanındaki tabloları görmek için,

```
mysql>SHOW TABLES;
+------------------+
| Tables_in_siniflar|
+------------------+
| 2a               |
+------------------+
1 row in set (0.00 sec)
```

komutunu kullanabilirsiniz. Bir tablodaki sütunları ve bunların özellikleri içinse şu komut gereklidir:

```
mysql> SHOW COLUMNS FROM 2a;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| no        | int(4)      |      | PRI | 0       |       |
| adisoyadi | varchar(50) | YES  |     | NULL    |       |
| fen       | int(2)      | YES  |     | NULL    |       |
| matematik | int(2)      | YES  |     | NULL    |       |
| sosyal    | int(2)      | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Veri girmek için **insert** komutu kullanılır. Bu komutun kullanımı şöyledir:

mysql> INSERT INTO <tabloAdı> (<sütunlar>) values (<değerler>)

```
mysql> INSERT INTO 2a (no, adisoyadi, fen,  matematik, sosyal)
    -> values (391, "Fatih Başer", 10, 10, 7);
Query OK, 1 row affected (0.05 sec)
```

Veri girişi yaparken şu iki noktaya dikkat etmeniz gerekiyor:

- Değerler, sütunların adını yazdığınız sırada olmalıdır. Sütunların adları istediğiniz herhangi bir sırada olabilir.
- Cümle değişkenler mutlaka tırnak içerisinde yazılmalıdır. MySQL'de tek tırnak (') ile çift tırnak (") arasında bir fark yoktur.

Bir tablodaki tüm kayıtları görmek için aşağıdaki komutu kullanabilirsiniz:

```
mysql> SELECT * FROM 2a;
+-----+-------------+------+-----------+--------+
| no  | adisoyadi   | fen  | matematik | sosyal |
+-----+-------------+------+-----------+--------+
| 391 | Fatih Başer |   10 |        10 |      7 |
| 291 | Hakan Kara  |    5 |         4 |      5 |
| 278 | Ufuk Dağcı  |    6 |         8 |     10 |
+-----+-------------+------+-----------+--------+
3 rows in set (0.00 sec)
```

Bu sorgulamadaki yıldız (*) tüm sütunları çağırmak anlamındadır. Eğer kaç tane kayıt (satır) olduğunu öğrenmek isterseniz:

```
mysql> SELECT COUNT(*) FROM 2a;
+----------+
| COUNT(*) |
+----------+
|        3|
+----------+
1 row in set (0.00 sec)
```

Özel sorgulamalarda yapabilirsiniz, örneğin adı "Fatih Başer" olan öğrencinin notlarını görmek için:

```
mysql> SELECT * FROM 2a WHERE adisoyadi="Fatih Başer";
+-----+-------------+------+-----------+--------+
| no  | adisoyadi   | fen  | matematik | sosyal |
+-----+-------------+------+-----------+--------+
| 391 | Fatih Başer |   10 |        10 |      7 |
+-----+-------------+------+-----------+--------+
1 row in set (0.00 sec)
```

Bu sorgulamada **2a** tablosunun tüm sütunları seçilecek. Fakat koşulumuz **adisoyadi** sütununun değerinin "Fatih Başer" olmasıdır. Eğer ismini hatırlamıyorsanız ve sadece soyadını biliyorsanız, şu şekilde bir sorgulama yapabilirsiniz:

```
mysql> SELECT no, adisoyadi, fen FROM 2a WHERE adisoyadi LIKE
"%Dağcı%";
+-----+------------+------+
| no  | adisoyadi  | fen  |
+-----+------------+------+
| 278 | Ufuk Dağcı |    6 |
+-----+------------+------+
1 row in set (0.00 sec)
```

Yukarıdaki sorgulamada **adisoyadi** sütunu içerisinde "Dağcı" geçen kelimeler üzerinde bir sorgulama yaptık. Yüzde işaretleri herhangi bir karakter kümesi anlamındadır. Diğer re'lere MySQL dökümanlarından bakabilirsiniz. Bu sefer tüm sütunları istemek yerine sadece **no, adisoyadi ve fen** sütunlarını istemiş olduk.

Belirli bir nottan daha büyük veya daha küçük olanları da sorgulayabiliriz. Örneğin fen dersinde 5'ten büyük not alanların listesini isteyelim:

```
mysql> SELECT no, adisoyadi, fen FROM 2a WHERE fen > 5;
+-----+-------------+------+
| no  | adisoyadi   | fen  |
+-----+-------------+------+
| 391 | Fatih Başer |   10 |
| 278 | Ufuk Dağcı  |    6 |
+-----+-------------+------+
2 rows in set (0.06 sec)
```

Fen dersi 5'ten küçük olanları istemiş olsaydık **fen<5** yazmamız gerekecekti.

Çapraz sorgulamalar da yapılabilir. Örneğin sosyal dersi 5'ten küçük veya fen dersi 5'ten büyük olanları isteyelim:

```
mysql> SELECT * FROM 2a WHERE sosyal < 5 OR fen > 5 ORDER BY no;
+-----+-------------+------+-----------+--------+
| no  | adisoyadi   | fen  | matematik | sosyal |
+-----+-------------+------+-----------+--------+
| 278 | Ufuk Dağcı  |    6 |         8 |     10 |
| 391 | Fatih Başer |   10 |        10 |      7 |
+-----+-------------+------+-----------+--------+
2 rows in set (0.00 sec)
```

Sorgulamaları Python karşılaştırma deyimlerine benzer şekilde yapabilirsiniz.
Herhangi bir sütunun değerini değiştirmek için update kullanılır. Şimdi 278 nolu öğrencinin sosyal dersini 5 yapalım:

```
mysql> UPDATE 2a SET sosyal=5 WHERE no=278;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Bir satırı tamamen silmek isterseniz **delete**'i kullanabilirsiniz:

```
mysql> DELETE FROM 2a WHERE adisoyadi LIKE "%Kara";
Query OK, 1 row affected (0.00 sec)
```

Tablo silme işlemi **drop table** ikilisi ile yapılır:

```
mysql> DROP TABLE 3b;
Query OK, 0 rows affected (0.00 sec)
```

İstediğiniz bir zamanda sütun ekleme şansına sahipsiniz. Eğer 2A sınıfına bir ders daha eklemek isterseniz, bu işlemi **table** ile yapmalısınız:

```
mysql> ALTER TABLE 2a ADD COLUMN resim INT(2);
Query OK, 2 rows affected (0.06 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Bu komut seti sadece sütun eklemek için değil aynı zamanda sütun silmek için de kullanılabilir. Şimdi eklediğimiz resim sütununu silelim:

```
mysql> ALTER TABLE 2a DROP COLUMN resim;
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

En uzun öğrenci isminin 50 değil de 37 olduğunu gördünüz. Bu durumda adisoyadi sütununun genişliğini 37 yapmak isterseniz:

```
mysql> ALTER TABLE 2a MODIFY adisoyadi VARCHAR(37);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Diğer değiştirme komut setleri için MySQL dökümanlarına bakınız.

Python-MySQL

Python içerisinden MySQL kullanmak oldukça kolaydır. Yazacağınız komut setleri yukarıda anlattıklarımızla aynıdır. **siniflar** veritabanına bağlanıp, **2a** tablosundaki tüm verileri alalım:

```
>>> import MySQLdb
>>> db=MySQLdb.Connection(host="localhost", user="mbaser", pass-
wd="yumurcak", db="siniflar")
>>> cursor=db.cursor()
>>> cursor.execute("SELECT * FROM 2a");
2L
>>> R=cursor.fetchall();
>>> cursor.close()
```

Şimdi **R** değişkeninin değerlerine bakalım:

```
>>> R
((391L, 'Fatih Ba\x9fer', 10L, 10L, 7L), (278L, 'Ufuk
Da\xa7c\x8d', 6L, 8L, 5L))
```

Gördüğünüz gibi, sonuçlar alınıp **R** değerine aktarılmış. **2a** tablosunda sadece 2 tane kayıt olduğundan:

```
>>> len(R)
2
```

R tüpünde de iki tane eleman vardır. Tüpün içerisinde ise tablodan alınan değerler vardır. Bu değerler, eğer hangi sütunların seçileceği belirtilmemişse, tablodaki sütunların sırasıyladır. Şimdi bu tüpü ekrana yazalım:

```
>>> for e in R:
...     for i in e:
...             print i,
...     print
...
391 Fatih Başer 10 10 7
278 Ufuk Dağcı 6 8 5
```

Her sorguda sadece bir elemanın yakalanmasını isteyebilirsiniz:

```
>>> db=MySQLdb.Connection(host="localhost", user="mbaser", pass-
wd="yumurcak", db="siniflar")
>>> cursor=db.cursor()
>>> cursor.execute("SELECT * FROM 2a");
2L
>>> R=cursor.fetchone();
>>> print R
(391L, 'Fatih Ba\x9fer', 10L, 10L, 7L)
>>> R=cursor.fetchone();
>>> print R
(278L, 'Ufuk Da\xa7c\x8d', 6L, 8L, 5L)
>>> R=cursor.fetchone();
>>> R
>>> cursor.close()
```

Bazen işaretçinin isteğiniz bir satıra gitmesini isteyebilirsiniz. O zaman **cursor.seek(<satırNo>)** kullanabilirsiniz. Hangi satırda olduğunu ise **cursor.tell()** ile öğrenebilirsiniz:

```
>>> cursor.seek(20)
>>> cursor.tell()
20
```

Eğer işaretçinin bulunduğu yerden, istediğiniz sayıda satır yakalamak istiyorsanız:

```
>>> R=cursor.fetchmany(5)
>>> len(R)
5
```

Bu iki işlem ile aslında tabloda istediğimiz iki aralığı almış olduk. Yani 20. satır ile 24. satır arasını yakalamış olduk. Bazen uzun veritabanlarında değerlerin ekrana bastırılmasında işimizi oldukça kolaylaştırmaktadır.

Peki bu işin Python'cası yok mu? Yani şöyle istediğimiz bilgileri sözlük halinde alsakta, alıştığımız şekilde kullansak olmaz mı? Bu soruları sor-

duğumuza göre cevabımız da hazır demektir. Python programcıları sözlükleri o kadar çok severler ki, sözlüksüz hiçbir şey yapmak istemezler, onlar Python'un tuzu biberidir. Şimdi imlecimizi sözlük cinsine alalım ve verileri yakalayalım:

```
>>> import MySQLdb
>>> db=MySQLdb.Connection(host="localhost", user="mbaser",
passwd="yumurcak", db="siniflar")
>>> cursor=db.cursor(MySQLdb.cursors.DictCursor)
>>> cursor.execute("SELECT * FROM 2a");
>>> R=cursor.fetchmany(2)
>>> R
({'adisoyadi': 'Fatih Ba\x9fer', 'fen': 10L, 'sosyal': 7L,
'matematik': 10L, 'no': 391L}, {'adisoyadi': 'Ufuk Da\xa7c\x8d',
'fen': 6L, 'sosyal': 5L, 'matematik': 8L, 'no': 278L})
```

İşte bu sonuç tam bize göre. Artık güzelce kullanabiliriz:

```
>>> print R[0]['no'], R[0]['adisoyadi'], R[0]['fen']
391 Fatih Başer 10
```

GUI



Python programlama dilinde grafik arabirimli programlar yazmak için elinizde birçok araç vardır. Bunlar gtk/gnome, qt/KDE, MFC/win32, wxPython ve Tkinter olarak sayılabilir. Başka araçlar varsa da pek ilgi görmezler. Python ile ön tanımlı olarak gelen araç ise Tkinter'dir. Eğer bir sistemde Python var ise mutlaka Tkinter de vardır. Ekstra paket kurmayı ve yapılandırma yapmayı gerektirmez. Oldukça hızlı ve boyut olarak küçük olması, GUI programcılığında kod yazılımını en alt düzeye indirmesi ve geliştirici sayısının çok olması nedeni ile hemen hemen tüm Python kitapları Tkinter'i anlatır. Biz de bu geleneği bozmayacak, sizlere mümkün olan en kolay GUI aracı Tkinter'den bahsedeceğiz.

Tkinter ilk olarak Tcl programlama dili için geliştirilmiş, daha sonra C/C++ kütüphaneleri de yazılmıştır. Son zamanlarda bu araca olan ilgi gittikçe artmakta ve geliştirici sayısı günden güne çoğalmaktadır. Yoğun olarak Tcl, Perl ve Python programcıları kullanmaktadır.

Tkinter oldukça yetenekli bir araçtır. Bir GUI aracından beklediğiniz her şeyi içerir. Kullandığınız Iddle Tkinter ile yazılmış olduğundan yeteneklerini az çok tahmin edersiniz.

Python 2.0'dan itibaren Tk8.3 kullanılmaktadır. Bu Türkçe karakterlerin görüntülenmesinde problem çıkarmaktadır. Python yazarı Guido van Russom ile yazışmamızda, bunun bir bug olduğunu söyledi ve buglist'e gönderildi. Henüz bir çözüm bulunamadığından Python 1.5.2 kullanmanızı tavsiye ederiz.



Pencere, Widget ve Olay

Pencere nesnelerin tanımlanabileceği alanı belirtir. Widget'lar ise pencereler üzerinde tanımlanan nesnelerdir. Tıpkı kağıt ve yazı gibi. Kağıt Pencereyi belirtirken, widget kağıt üzerindeki yazı ve resimleri ifade eder. Bu iki kavramı birbirinden iyice ayırmanız gerekir.

Bilgisayarınızın masaüstünü düşünün, masaüstü bir pencere, oun üzerindeki ikonlar da birer widget'tir. Her ne kadar pencerenin çerçevesi çıkmasa da aslında bir Explorer penceresinden başka bir şey değildir masaüstünüz.

Web sayfalarındaki düğmeler, yazı yazılabilecek alanlar, seçim yapabileceğiniz radyo düğmeleri... Hepsi birer widget'tir. Aslında sayfa üzerindeki yazılar bile birer widget'tir.

Bir GUI programı başlatıldığında, ilk yapılan işlerin dışında kullanıcı ona bir şey yapmasını söyleyene kadar bekler. Bu ancak bir döngü ile gerçekleşebilir. Önceki örneklerimize bir göz atalım:

```
R=''
while not R in ['''E,'H']:
R=raw_input('[E]vet/[H]ayır ')
```

Bu programda bir döngü başlatılıyor, kullanıcı herhangi bir tuşa bastığında yaptığı iş değerlendiriliyor ve E ya da H tuşuna basana kadar döngü devam ediyor. GUI programları da buna benzer şekilde çalışmaktadır. Kullanıcının ona bir tepki vermesi gerekiyor. Bu tepkiye GUI Programcılığında "**olay**" denir. Olay herhangi bir tuşa basmak, fareyi hareket ettirmek, farenin tuşlarından birisine tek tıklamak ve çift tıklamak gibi, kullanıcının verdiği tepkilerdir. Web sayfalarında, bir linke tıklamak, arama butonuna basmak ya da fareyi hareket ettirmek hep birer olaydır.

Merhaba GUI

Programlama dili kitaplarında alışılagelmiş "Merhaba" lafını kitabımızın en başında kullanmadık çünkü onu buraya sakladık. Şimdi bir pencere açacağız ve üzerine "Merhaba GUI" etiketini yazacağız:

program16-01

```
#!/usr/bin/python

from Tkinter import *
anaPencere=Tk()
widget = Label(anaPencere)
widget.config(text="Merhaba GUI")
widget.pack()
anaPencere.mainloop()
```

Bu kodu yazıp çalıştırdığımızda, ekranınızda bir pencere görünecek ve içerisinde "Merhaba GUI" yazacaktır, tıpkı **Şekil 16.1**'de olduğu gibi. Etkileyici değil mi?



Şekil 16.1: Pyhton'da ilk GUI programımız

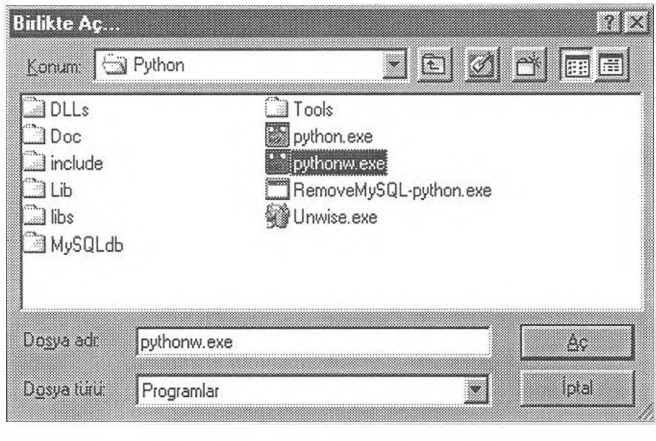
Yazdığımız programı satır satır inceleyelim:

İlk satırın ne olduğunu biliyorsunuz: Bu, programın kimin tarafından yürütüleceğini belirtiyor. Sonraki satırda ise **Tkinter** modülünden tüm nesnelerini çağırdık. **Tk()** ana penceresini **anaPencere** değişkenine atadık. Daha sonra **anaPencere**'de bir etiket (label widget'i oluşturduk. Oluşturduğumuz etiketin üzerinde görünecek metni, **config()** fonksiyonu ile yazdık. Oluşturulan widget'lerin pencerede görünebilmesi için bir **geometri yöneticisi**'ne ihtiyacımız vardır. En basit ve kolay kullanılan **pack()**'dir. Son olarak **anaPencere** ana döngüsünü başlattık.

Bu basit bir Tkinter programıydı, fakat diğer tüm programlara ışık tutmaktadır. Yazılacak en karmaşık programlarda da yine aynı işlemler yapılır. Tüm Tkinter GUI programlarında mutlaka şunlar yapılır:

- ana pencerenin oluşturulması
- widget'lerin tanımlanması ve yapılandırılması
- widget'lerin yapıştırılması
- ana döngünün başlatılması.

Windows kullanıcıları bu programı yazıp kaydettikten sonra üzerine çift tıklayarak çalıştırırlarsa, önce bir DOS penceresinin açıldığını göreceklerdir. Daha sonra asıl prgoram çalışacaktır. DOS penceresi Python yorumlayıcısı tarafından başlatılır ve sonra programınız yorumlanır. Bu pencere herkesi rahatsız eder. Görmek istemiyorsanız (eminim istemezsiniz), programınızın uzantısını **.py** yerine **.pyw** yapın. Yani GUI programlarınızın uzantısı **.pyw** olacaktır. Programınızı çalıştıramıyorsanız, ayarlarınız eksik yapılmış demektir. Bunun için herhangi bir **.pyw** uzantılı programın üzerinde farenizin sağ tuşuna tıklayın, daha sonra "Birlikte Aç" seçeneğini seçin, karşınıza çıkan pencerede diğer seçeneğini seçerek, Python'u kurduğunuz patikadan **pythonw.exe** programınızı seçin, **Şekil 16-2**'de görüyorsunuz. "Aç" dedikten sonra, "Bu dosya türünü açmak için her zaman bu programı kullan" seçeneğini tıklamayı unutmayın.



Şekil 16-2: GUI Programlarının pythonw.exe ile çalışmasını bu şekilde sağlayın.

Eğer ilk programınızı yazıp çalıştırırsanız ve pencere boyutunu sağ alt köşteden tutup çekerek büyüttüyseniz, "Merhaba GUI" etiketinin pencerenin ilk başında kaldığını göreceksiniz. Çünkü biz bu etiketi pencereye yapıştırırken herhangi bir parametre vermedik. Etiket de ön tanımlı olarak en üste yapıştırıldı. Widget'leri yapıştırırken vereceğiniz parametreler sayesinde, mümkün olan tüm alanları kaplamasını veya belirli bir büyüklükte olmasını sağlayabilirsiniz. İşte aşağıda, etiketimizin pencerenin tamamını kaplaması için yapılması gerekenler:

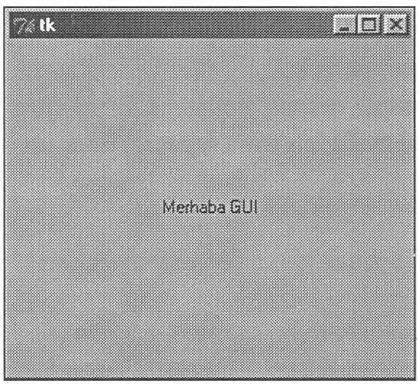
program16-02

```
#!/usr/bin/python

from Tkinter import *
```

```
anaPencere=Tk()
widget = Label(anaPencere)
widget.config(text="Merhaba GUI")
widget.pack(expand=YES, fill=BOTH)
anaPencere.mainloop()
```

Programı çalıştıralım ve ekran boyunu fare ile biraz büyütelim. Sonucu **Şekil 16.3**'te görüyorsunuz.



Şekil 16.3: Widget'in pencerenin tamamını kaplaması.

Pencere Başlıkları

Yukarıdaki programları çalıştırdığınızda pencere başlığının hep "Tk" olduğunu görürsünüz. Bunu değiştirmek için, ana pencerenin **title()** fonksiyonunu kullanmak gerekiyor:

program16-03

```
#!/usr/bin/python

from Tkinter import *
anaPencere=Tk()
anaPencere.title("Python Programı")
widget = Label (anaPencere)
widget.config(text="Merhaba GUI")
widget.pack()
anaPencere.mainloop()
```

Bu programı çalıştırırsanız, **Şekil 16.4**'te görüldüğü gibi pencerenin başlığının değiştiğini görürsünüz.



Şekil 16.4: Pencere başlığının değiştirilmesi.

Düğmeler ve Geri Çağırımlar

GUI programcılığının ana teması bir yere tıklayarak tanımlanan işin yapılmasıdır. Bu amaçla çeşitli widget'ler vardır. Bunların en çok kullanılanı ise düğmelerdir. Düğmeler tüm programların en çok kullanılan widget'lerdir. Tkinter ile düğme oluşturmak için **button()** nesnesi kullanılır. Bir düğme oluşturulduğuna göre, o düğmeye tıklandığında, mutlaka yapacağı bir iş vardır. Buna **geri çağırım** diyoruz.

Aşağıdaki program basit bir düğme oluşturmak için neler yapmanız gerektiğini açıklıyor:

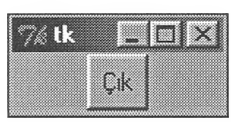
program16-04

```
#!/usr/bin/python

import sys
from Tkinter import *

ana=Tk()
widget=Button()
widget.config(text="Çık")
widget.config(command=sys.exit)
widget.pack()
ana.mainloop()
```

Bu programda "Çık" isimli bir düğme tanımlandı, üzerine tıklandığında programı sonlandıracak olan **sys.exit** çağrıldı. Örnek pencereyi **Şekil-16.7**'de görüyorsunuz:

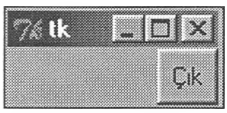


Şekil-16.5: Bir düğme ve geri çağırımı: Çık

Bundan sonraki programlarımızda yazılımı kısaltmak amacı ile anaPencere yerine sadece ana ismini kullanacağız.



Yukarıdaki programda düğmenin ortaya yerleştiğini görüyorsunuz, peki bunu sol ya da sağ tarafa alabilir miyiz? Evet, yapmanız gereken yapıştırma işleminde bu widget'in hangi tarafa yapışacağını belirtmek olacaktır. Yukarıdaki programda **widget.pack()** yerine **widget.pack(side=RIGHT)** yazarsanız, pencere **Şekil-16.6**'daki gibi olacaktır. Right yerine left yazarsanız, düğme sol tarafa yapışır.



Şekil-16.6: Düğmenin sağ tarafa alınması.

Düğmeler tıpkı, etiketler gibi genişleyebilirler. Etiketleri yapıştırırken kullandığımız **expand=YES** ve **fill=?** parametresi düğmelerde de kullanılabilir. Fill parametresinin alabileceği değerler:

X: Ekran büyüdüğünde, büyüyebilir bir widget ise (expand=yes ise), yatay yönde büyüme sağlanır.

Y: Ekran büyüdüğünde, büyüyebilir widget ise, dikey yönde büyüme sağlanır.

BOTH: Büyüebilir widget'ler ekran büyüdüğünde her iki yönde büyüyebilir. Şimdi yukarıdaki programı biraz değiştirelim:

program16-05

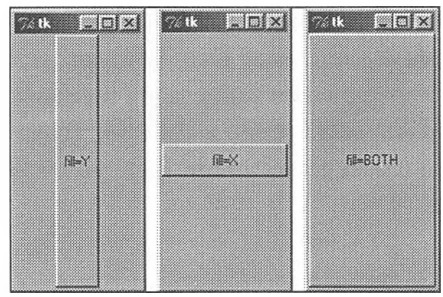
```
#!/usr/bin/python

import sys
from Tkinter import *

ana=Tk()
widget=Button()
widget.config(text="fill=X")
widget.config(command=sys.exit)
```

```
widget.pack(expand=YES, fill=BOTH)
ana.mainloop()
```

Şekil 16.7'de fill'in alacağı parametrelere göre pencereler görünüyor.



Şekil 16.7: Fill'in alabileceği parametrelere göre düğmenin büyümesi.

Şimdi bizim tanımlayacağımız bir fonksiyonu, oluşturduğumuz bir düğme ile çağıralım. Aşağıdaki programda "**Şarkılar**" düğmesine tıklandığında **sarkilar** listesinde bulunan şarkı isimlerinden rastgele birisi, düğme metni olarak görünecektir.

program16-06

```
#!/usr/bin/python

import random
from Tkinter import *

sarkilar=['Seni Seviyorum','Gülüm Benim', 'Bal Dudaklım','Kurbanın
Olayım']

def degistir():
    widget.config(text=random.choice(sarkilar))

ana=Tk()
widget=Button()
widget.config(text="Şarkılar")
widget.config(command=degistir)
widget.pack(expand=YES, fill=BOTH)
ana.mainloop()
```

Yukarıdaki programda çok basit bir geri çağırım yapıldı. Normalde bir düğme oluşturulduğunda üzerine bir kere tıklanır ve bir fonksiyon çağrılır. Yukarıdaki açıklamalarımıza göre bu olası olaylardan birisidir. Düğmenin üzerine sağ tuşla tıkladığımızda başka bir olay ya da sol tuşa çift tıklandığında başka bir olay gerçekleşir. Bu olayların her birisinde farklı bir fonksiyonun çağırılması istenebilir. O halde bir düğmeye farklı geri çağırımlar atamamız gerekmektedir. Bu atama işlemine **kayıt yaptırma** denir. İşlem, **bind()** fonksiyonu ile gerçekleştirilir.

Şimdi bir düğme oluşturalım; kullanıcı sağ tuşa tıkladığında bir fonksiyon, sol tuşa tıkladığında başka bir fonksiyon geri çağırılsın:

program16-07

```
#!/usr/bin/python

from Tkinter import *

def solTus(event):
    widget.config(text="Sol tuşa tıkladınız")

def sagTus(event):
    widget.config(text="Sağ tuşa tıkladınız")

ana=Tk()
widget=Button()
widget.config(text="Sol ya da sağ tuşa tıklayınız")
widget.bind('<Button-1>', solTus)
widget.bind('<Button-3>', sagTus)
widget.pack()
ana.mainloop()
```

Bu programı çalıştıralım, önce sol sonra sağ tuşa tıklayalım. Karşılaşacağınız ekranlar sırası ile **Şekil-16.8**'de gösterilmiştir. Kayıt yapılırken olaylar **"<olay>"** şeklinde bir cümle halinde yazılır.



Şekil-16.8: Program çalıştıktan sonra kullanıcı önce sol sonra sağ tuşa tıklanıyor.

Yukarıdaki programlarda geri çağrım yapılan fonksiyonların **event** isimli birer argüman aldığını görüyorsunuz. Bu argümanın, sınıfları anlatırken kullandığımız **self**'e benzer bir görevi olduğunu söylemek sanırım aydınlatıcı olmuştur.

Geri Çağrımlara Veri Aktarımı

Şimdiye kadar verdiğimiz örneklerde geri çağrımlara hiç veri aktarmadık, oysa her düğme için bir fonksiyon yazmak pek akıllıca olmayabilir. Yukarıdaki örneklerde geri çağrım yapılan fonksiyonların yazılımında parantezleri kullanmadık, genelde fonksiyonlar çağrılırken

fonksiyon_adi(argüman)

şeklinde yazılır. Bununla birlikte biz **command** argümanına sadece fonksiyonun adını verdik. Argüman belirtme şansımız yok. Bunu çözmek için, ilk bölümlerde öğrendiğimiz **lambda** fonksiyonunu kullanacağız. Aşağıdaki örnekte bunun nasıl yapılacağı görülüyor:

program16-07b

```
#!/usr/bin/python

from Tkinter import *

def DugmeX(x):
    L.config(text="Şimdi '%s' düğmesine tıkladınız." % x)

ana=Tk()
L=Label(text="Aşağıdaki düğmelerden birisine tıklayınız")
L.pack()

B1=Button(text="Çıkma")
B1.config(command=(lambda : DugmeX("Çıkma")))
B1.pack(side="left")
```

```
B2=Button(text="Çık")
B2.config(command=(lambda : DugmeX("Çık")))
B2.pack(side="right")

ana.mainloop()
```

Tkinter Olayları

En son yazdığımız programda kullanıcının verdiği tepkiyi ayırt etmiştik. Programımız sol ya da sağ tuşa tıklandığını ayırt edebiliyor. Kullanıcının verdiği tepkileri olay diye adlandırmıştık. Tkinter'da çok çeşitli olaylar vardır. Bunların tam listesi ve açıklamaları aşağıdaki tabloda verilmiştir.

Olay	Açıklaması
Activate	Bu pencere/widget şu anda aktif.
ButtonPress, Button	Fare tuşuna tıklama; Button-1=sol tuş, Button-2=orta tuş, Button-3 sağ tuş, Double Button-1=sol tuşa çift tıklama...
ButtonRelease	Fare tuşunu bırakma
Circulate	Pencerenin/widget'in z-yönünde değiştirilmesi.
Configure	Boyut, şekil veya yerinin değiştirilmesi.
Deactivate	Bu pencere/widget artık aktif değil.
Destroy	Bu widget yok edilmek üzere.
Enter	Fare işaretçisi bu widget üzerine gelmiştir.
Leave	Fare işaretçisinin widget üzerinden ayrılması.
Expose	Bu pencere/widget'in bazı veya hepsi açığa çıkmayı değiştirmiştir.
FocusIn	Widget'in klavye odağı vardır.
FocusOut	Widget klavye odağını kaybetmiştir.
KeyPress, Key	Bir tuşa basma: Keya=klavyenin a tuşu...
KeyRelease	Basılan tuşun geri bırakılması.
Motion	Fare işaretçisinin hareket etmesi.
Map	Bu pencere/widget görüntülendi.
Property	Pencerenin/widget'in bazı özellikleri değişti, örneğin rengi.
Reparent	Ana pencere/widget başka bir pencere/widget'e değişti.
Unmap	Bu pencere/widget görüntülenmedi.
Visibility	Görünürlük değişti; küçültüldü veya büyültüldü.

Bu olaylarla ilgili örnekler yazmamız neredeyse imkansız. Sadece farenin düğme üzerine getirilmesi ve dışına çıkartılması ile ilgili bir örnek vererek olaylar ve kayıt yapma konusunu kapatalım:

program16-08

```
#!/usr/bin/python

import sys
from Tkinter import *

def gel(event):
    widget.config(text="Heeey çek üzerimden şu fareyi")

def git(event):
    widget.config(text="Oh be fare üzerimden gitti...")

ana=Tk()
widget=Button()
widget.config(text="Fareyi üzerime getir")
widget.bind('<Enter>', gel)
widget.bind('<Leave>', git)
widget.pack()
ana.mainloop()
```

Çoklu Widget'ler

Bir pencere üzerine sadece bir düğme değil çeşitli widget'ler eklenir. Şimdi, penceremiz üzerine üç tane widget koyalım. Bunlardan ilki bir etiket olsun. Diğeri tıklandığında etiketin metnini değiştiren bir düğme ve sonuncusu programdan çıkış düğmesi olsun:

program16-09

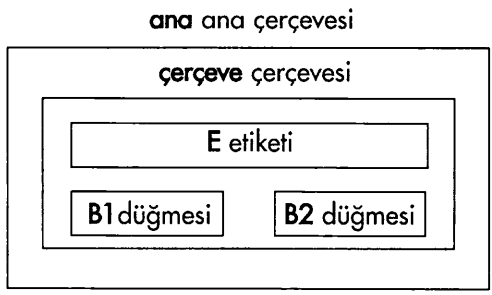
```
#!/usr/bin/python

from sys import exit
from Tkinter import *
```

```
def tikla():
    E.config(text="Tıklama demiştim sana")

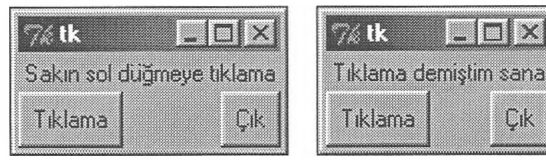
ana=Tk()
cerceve=Frame()
cerceve.pack()
E=Label(cerceve)
E.config(text="Sakın sol düğmeye tıklama")
E.pack(side=TOP)
D1=Button(cerceve)
D1.config(text="Tıklama")
D1.config(command=tikla)
D1.pack(side=LEFT)
D2=Button(cerceve)
D2.config(text="Çık")
D2.config(command=exit)
D2.pack(side=RIGHT)
ana.mainloop()
```

Yukarıdaki program diğerlerine göre biraz farklı. Şimdiye kadar ana penceremizi üzerine nesnelerin koyulabileceği alanlara ayırmamıştık. Dolayısıyla tüm nesneler (widget'ler) ana pencerenin birer üyesiydi. Daha doğrusu, widgetlerin ebeveyni hep ana pencereydi. Şimdi **cerceve** isimli ile bir çerçevemiz var. Bir widget oluşturduğumuzda bunun ebeveyninin kim olduğunu da belirtmek durumundayız, aksi halde ana pencere olur. Bu programda **cerceve**'nin ebeveyni **ana**, etiket ve düğmelerin ebeveyni de **cerceve**'dir. **Şekil 16.9**'de bu ilişkiyi daha iyi görebilirsiniz:



Şekil-16.9: Ana pencere ve çocuklarının yerleşimi.

Yukarıdaki programı çalıştırdığımızda, karşılaşacağımız ekran Şekil 16.10'daki gibi olacaktır.



Şekil-16.10: Çoklu widget'lerin bir çerçeveye yerleştirilmesi.

Şimdi isterseniz yapıştırma işlemlerine biz göz atalım. Öncelikle bir Label widget'imiz var, bu çerçevenin en üst kısmına side=TOP argümanı ile yapıştırıldı. Daha sonra "Tıklama" düğmesi side=LEFT ile geri kalan alanın sol kısmına yapıştırıldı. "Çık" düğmesi ise en son kısmın sağ tarafına side=RIGHT ile yapıştırılmış oldu.

Yazı Tipleri ve Renkleri

Tkinter pencerelerindeki yazı tiplerini istediğiniz gibi değiştirebilirsiniz. Onlara renkler verebilir, kalınlaştırıp yatık hale getirebilirsiniz. Burada anlatacağımız yazıtipi işlemleri, metin yazılabilecek tüm widget'larda geçerlidir. Yazıtipi ve şeklini değiştirmek için yine **config()** fonksiyonundan yararlanırız. Aşağıdaki programı yazıp çalıştıralım:

program16-10

```
#!/usr/bin/python

from sys import exit
from Tkinter import *

ana=Tk()
E=Label()
E.config(text='Harika Python')
E.config(font=("Comic Sans MS", 30, "bold italic underline"))
E.config(bg="orange", fg="red")
E.pack()
ana.mainloop()
```

Bu programda bir widget için birden fazla **config()** fonksiyonu kullandık. İlk yaptığımız iş, etiketin üzerine yazılacak metni yapılandırmak, daha sonra metnin yazı tipini, puntoyu ve yazı tipi şeklini belirtmek oldu. Son olarak etiketin arka plan ve metin rengini ayarlamak oldu.

Bu programı yazıp çalıştırırsanız ve sisteminizde Comic Sans MS yazıtipi yüklüyse **Şekil 16.11**'deki gibi bir pencereye sahip olursunuz:



Şekil-16.11: Widget'lerde yazı tipi ve rengin belirtilmesi

Yazı Tiplerini Belirtmek

Tkinter'da yazı tipi belirtirken sisteminiz tarafından kullanılabilen tiplerden birisini seçmelisiniz. Aksi takdirde ön tanımlı yazı tipine düşecektir. Windows kullanıyorsanız ekleyebileceğiniz yazıtiplerini herhangi bir Windows uygulamasından veya "Ayarlar / Denetim Masası/ Yazı tipleri" yolundaki program ile görebilirsiniz. UN*X kullanıyorsanız **xfontsel** prgoramı ile öğrenebilirsiniz. Daha sonra harflerin boyutunu punto cinsinden yazmalısınız. Son olarak yazı tipi şeklini belirtmeniz gerekiyor. Yazı tipi şekli şunlardan birisi veya aralarına bir boşluk koymak kaydıyla herhangi bir kombinasyonu olabilir:

bold: Kalın
italic: Yan yatık
underline: Altı çizgili

UN*X tipi işletim sistemlerinde yazı tipleri ve dil çözünürlükleri şu şekilde belirtilebilir:

Renk Belirtmek

Metinlerin renklerini belirtmek için **bg** (arkaplan) ve **fg** (metin) argümanlarını **config()** fonksiyonuna verebilirsiniz. Bu argümanların alabileceği değerler için ya renklerin ingilizce karşılıklarını ya da html sayfalarında kullanılan hexadecimal renk kodlamasını kullanabilirsiniz.

Renk ve yazı tipi belirtimi sadece etiketlerde değil, üzerine yazı yazdığınız herhangi bir widget'e de uygulanabilir. Aşağıda değişik renk ve yazı tiplerine sahip iki düğme görüyorsunuz:

program16-11

```
#!/usr/bin/python

from sys import exit
from Tkinter import *

ana=Tk()
cerceve=Frame()
cerceve.pack()

D1=Button(cerceve)
D1.config(text="Birinci düğme")
D1.config(font=("Verdana", 20, "bold"))
D1.config(bg="#cadeab", fg="blue")
D1.pack(side=LEFT)

D2=Button(cerceve)
D2.config(text="İkinci Düğme")
D2.config(font=("Courier New", 15, "italic underline"))
D2.pack(side=RIGHT)

ana.mainloop()
```

Bu program **Şekil 16.12**'deki pencereyi verir.



Şekil 16.12: Metin yazılan her yerde yazı tipi ve şeklini belirtebilirsiniz. Renk ise her widget'a uygulanabilir.

Widget Boyutları

Widget'ların belirli büyüklükte olmasını istediğinizde, onları **height** (boy) ve **width** (en) argümanları ile birlikte **config()** fonksiyonuna verebilirsiniz. Aşağıda sol taraftaki düğmenin boyu ve eni değiştirilmiş şekilde görünüyor.

program16-12

```
#!/usr/bin/python

from sys import exit
from Tkinter import *

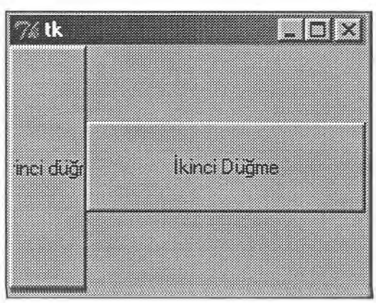
ana=Tk()
cerceve=Frame()
cerceve.pack()

D1=Button(cerceve)
D1.config(text="Birinci düğme")
D1.config(height=10, width=5)
D1.pack(side=LEFT)

D2=Button(cerceve)
D2.config(text="İkinci Düğme")
D2.config(width=25, height=3)
D2.pack(side=RIGHT)

ana.mainloop()
```

Program çalıştırıldığında aynen **Şekil-16.13**'de olduğu gibi görünüyor.



Şekil-16.13: Widget'ların eni ve boyu değiştirilebilir.

Widget Görüntülerini Değiştirmek

Widget'lar etrafına çerçeve koymak için **bd=N**, bunun tipini belirtmek için ise **relief** argümanını **config()** fonksiyonuna verebilirsiniz. Relief'in alabileceği değerler şunlar: FLAT, SUNKEN, RAISED, GROOVE, SOLID, RIDGE.

Aşağıdaki programda etiket üzerine yazılmış metnin etrafına bir çerçeve koyduk:

program16-13

```
#!/usr/bin/python

from sys import exit
from Tkinter import *

ana=Tk()
El=Label(text="Bu yazının etrafı çerçevelidir")
El.config(bd=3, relief=SOLID)
El.pack(pady=5)

E2=Label(text="Bu yazının etrafı içe göçük çerçevelidir")
E2.config(relief=GROOVE)
E2.pack(pady=5)

E3=Label(text="Bu yazının etrafı yukarı kalkık çerçevelidir")
E3.config(bd=3, relief=RIDGE)
E3.pack(pady=5)

E4=Label(text="Bu yazı içe çöküktür")
E4.config(bd=4, relief=SUNKEN)
E4.pack(pady=5)

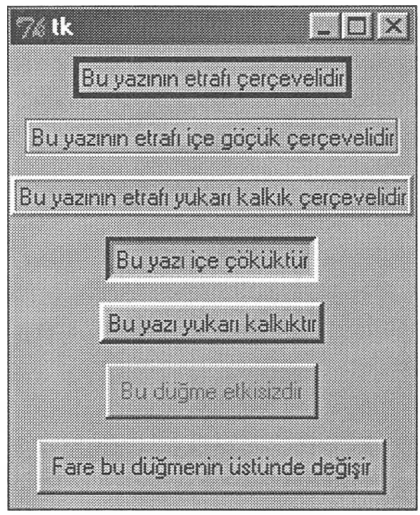
E5=Label(text="Bu yazı yukarı kalkıktır")
E5.config(bd=3, relief=RAISED)
E5.pack(pady=5)
Bl=Button(text="Bu düğme etkisizdir")
```

```
B1.config(state=DISABLED)
B1.pack(pady=5)

B2=Button(text="Fare bu düğmenin üstünde değişir")
B2.config(cursor="hand2")
B2.pack(pady=5)

ana.mainloop()
```

Program çalıştırıldığında **Şekil 16.14**'teki gibi bir görüntüyü vermektedir.



Şekil-16.14: Widget'ler üzerinde çeşitli etkiler.

Çerçeveler

Widgetların etrafına çerçeve koymak için **config()** fonksiyonuna **bd** parametresini vermeniz gerekiyor. Bu parametrenin alacağı tamsayı değerler çerçevenin pixel cinsinden kalınlığını ifade edecektir. Çerçeve kalınlığını verdikten sonra mutlaka, hangi cinsten bir çerçeve olacağını **relief** parametresi ile belirtmeniz gerekecektir. **relief**'in alabileceği değerler FLAT, SUNKEN, RAISED, GROOVE, SOLID veya RIDGE olabilir. Bunların etkilerini anlatmayacağız çünkü yukarıda verdiğimiz örnekte hepsinin etkisi görünmektedir. **relief**'e değer atadığınız halde çerçeve genişliğini belirtmezseniz, kalınlık 1 piksel olur.

Durum

Windows (veya X-Windows) kullanıcısı iseniz, bazen menülerin bir kısmının veya seçeneklerin bir kısmının etkin olmadığını görürsünüz. Programcı programını yazarken o seçenekleri kullanmak için şartların oluşup oluşmadığını denetler, menüler ekranda göründüğü halde seçilemezler ve üzerine tıklanamazlar. Bunu Tkinter'de yapmak için **state** parametresine değer atamalısınız. Seçenekleri etkisiz hale getirmek istiyorsanız DISABLED, aktif yapmak için de NORMAL değerlerini kullanmanız gerekir.

Yerleşke

Widget'leri istediğiniz şekilde yerleştirebilirsiniz, bunu belirtmek için **padx** ve **pady** parametreleri kullanılır. Bu parametreler iki widget arasında bırakılacak mesafeyi piksel cinsinden belirtir. **x** yatay mesafe; **y** ise dikey mesafedir.

İşaretçi

Fare işaretçisinin bir alan ya da widget üzerinde başka bir şekle bürünmesi için cursor parametresine şu değerlerinden birsini atayabilirsiniz: "watch", "pencil", "cross", "hand1" veya "hand2". Diğer işaretçi çeşitleri için Tk dokümanlarına bakmalısınız.

Üst Düzey Pencereler

Bir GUI programı bazen birden çok pencereden oluşabilir. Aynı sürece bağlı diğer pencerelere üst-düzey pencereler denir. Bunlar her zaman ana pencereye bağlıdırlar. Aşağıda yazacağımız örnek program, ana pencerenin yanı sıra bir tane de üst-düzey pencere açacaktır. Ana penceredeki düğmeye her tıkladığınızda yeni bir üst-düzey pencere açılacaktır.

program16-14

```
#!/usr/bin/python

from Tkinter import *

def pencereAc():
    Penceren=Toplevel()
    Penceren.title("Penceren")
```

```
    BP2=Label(Penceren)
    BP2.config(text="Sonradan Açılan üst,düzey pencere")
    BP2.pack()

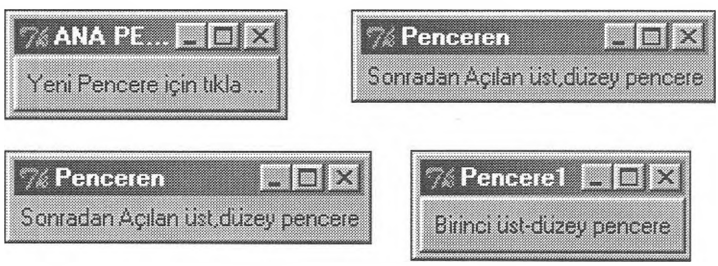
ana=Tk()
ana.title("ANA PENCERE")
Pencere1=Toplevel()
Pencere1.title("Pencere1")

B=Button(ana)
B.config(text="Yeni Pencere için tıkla ...")
B.config(command=pencereAc)
B.pack()

BP1=Button(Pencere1)
BP1.config(text="Birinci üst düzey pencere")
BP1.pack()

ana.mainloop()
```

Programın verdiği pencerelerin örneği **Şekil-16.15**'te görünüyor.



Şekil-16.15: Ana pencere ve buna bağlı düğer üst düzey pencereler.

Diyaloglar

Bir GUI programı kullanıcı ile etkileşime geçeceğine göre, ara sıra kullanıcının yanlış yaptığı işleri ona iletme veya yapacağı işin onayını alma gibi diyaloglar gerçekleşir. Veya birtakım verilerin girilmesi, renk seçimi, dosya

seçimi gibi standart diyaloglar vardır. Bir programcının ihtiyaç duyabileceği her türlü diyalog Tkinter'da mevcuttur.

Bilgilendirme Diyalogları

Bilgilendirme diyalogları dört çeşittir. Bunlar kullanıcıyı uyarma, yapılan işin onaylanması için cevap bekleme veya iş bittiğinde kullanıcıya bilgi verme şeklindedir. Aşağıdaki örnek tüm bu dört diyaloğu içeriyor.

program16-15

```
#!/usr/bin/python

from Tkinter import *
from tkMessageBox import *

def onayAl():

    if askyesno('Onayla', 'Hard diskinizi biçimlemek istediğinizden
emin misiniz?'):
        showwarning('Evet', 'Hard diskinizi tertemiz yaptım.')
    else:
        showinfo('Hayır', "Biçimleme iptal edildi.")

def hata():
    showerror("Hata!","Bu programdan çıkmak yasaklanmıştır.")

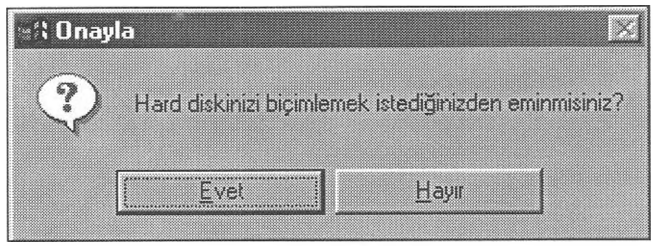
ana=Tk()

B1=Button()
B1.config(text="Biçimle")
B1.config(command=onayAl)
B1.pack(side=LEFT)

B2=Button()
B2.config(text="Çık")
B2.config(command=hata)
B2.pack(side=RIGHT)

ana.mainloop()
```

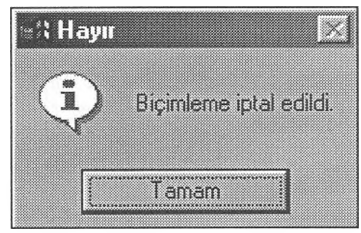
Programın verdiği örnek diyalog görüntüleri **Şekil-16-19** arasında gösterilmiştir.



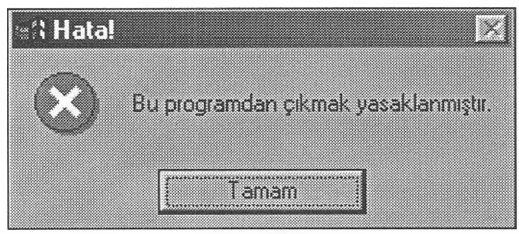
Şekil-16.16: Onaylama diyaloğu.



Şekil-16.17: Uyarı diyaloğu.



Şekil- 16.18: Bilgilendirme diyaloğu.



Şekil-16.19: Hata diyaloğu.

Dosya Açma Diyaloğu

Kullanıcının bir dosya seçmesi için kullanılan diyalogdur. Bunu **tkFileDialog** modülünde bulunan **askopenfilename()** fonksiyonu ile yapabilirsiniz. Aşağıdaki program bir dosya seçimi sonlanınca etikete seçilen dosyanın tam patikasını yazmaktadır.

program16-16

#!/usr/bin/python

```
from Tkinter import *
```

Python

```
from tkFileDialog import *

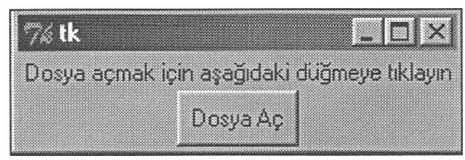
def dosyaAc():
    a=askopenfilename()
    if a: E.config(text=a)
    else: E.config(text="Dosya açma işlemi iptal edildi.")

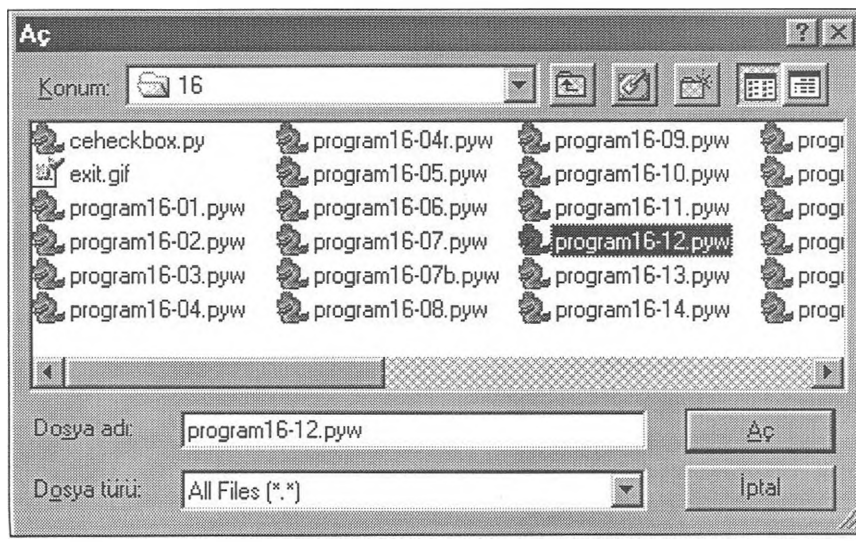
ana=Tk()

E=Label(text="Dosya açmak için aşağıdaki düğmeye tıklayın")
E.pack()
D=Button(text="Dosya Aç")
D.config(command=dosyaAc)
D.pack()

ana.mainloop()
```

Bu program **Şekil-16.20**'deki gibi kullanıcının bir dosya açmasını isteyecektir.







Şekil-16.20: Dosya açma diyaloğu

Renk Seçim Diyaloğu

Bu diyalog **tkColorChooser** modülünde bulunan **askcolor** ile çağrılabilir. Aşağıdaki program, düğmeye tıklandığında kullanıcının bir renk seçmesini istiyor ve seçilen renk, etiket üzerinde bulunan metnin arka plan rengi olarak ayarlanıyor.

program16-17

```
#!/usr/bin/python

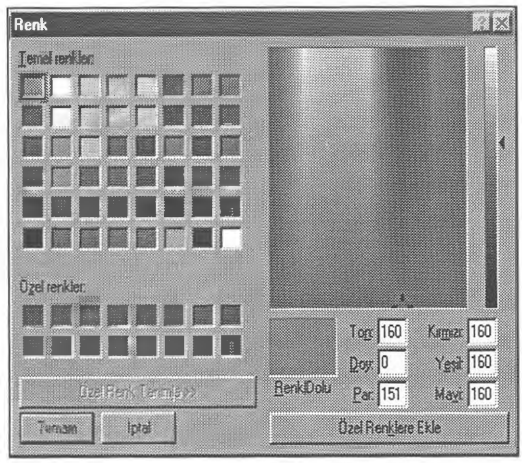
from Tkinter import *
from tkColorChooser import *

def renkSec():
    a=askcolor()
    E.config(bg=a[1])

ana=Tk()

E=Label(text="Arkaplan rengini değiştirmek\n için düğmeye tıkla-
yınız")
E.pack()
D=Button(text="Renk Seç")
D.config(command=renkSec)
D.pack()

ana.mainloop()
```



Şekil-16.21:
Tkinter ile çağrılan renk seçim diyaloğu. Renk seçim diyaloğu bir tüp şeklinde iki değer gönderir. Bunlardan ilki yine bir tüp şeklinde RGB renk kodları ve ikincisi ise hexadecimal cinsinden koddur.

Bu program çalıştırılıp "Renk Seç" düğmesine tıklandığında **Şekil-16.21**'deki diyalog açılıyor.

Entry Widget'i

Kullanıcılardan bir satır halinde bilgi girişi isteniyorsa bu **entry** widget'i ile yapılır. Girilen metin, widget'in atandığı nesnenin **get()** fonksiyonu ile alınabilir. Aşağıdaki örnekte bir bilgi girişi isteniyor. Bu bilgi etikete yazılır:

program16-18

```
#!/usr/bin/python

from Tkinter import *

def yaz():
    E.config(text= "Merhaba %s" % Bgir.get())

ana=Tk()

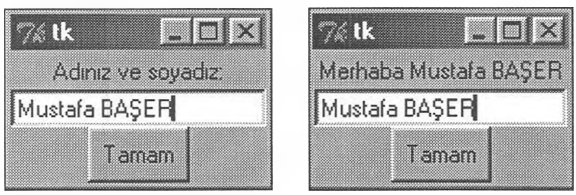
E=Label(text="Adınız ve soyadız:")
E.pack()

Bgir=Entry()
```

```
Bgir.pack()

D=Button(text="Tamam")
D.config(command=yaz)
D.pack()
ana.mainloop()
```

Bu programın örnek çalışması **Şekil-16.22**'de gösterilmiştir.



Şekil-16.22: Bilgi girişi örneği.

Message Widget'i

Önceki programlarımızda ekran üzerindeki açıklamaları **label** widget'i ile yapmıştık. **label** genelde kısa metinler içindir. Eğer metin çok uzun olacaksa en iyisi **message()** widget'ini kullanmak olacaktır. Böylelikle metni istediğiniz gibi sağa, sola yaslayabilir veya ortalayabilirsiniz. Aşağıdaki program ekran üzerine uzun bir açıklama metni yazdırıyor:

program16-19

```
#!/usr/bin/python

import sys
from Tkinter import *

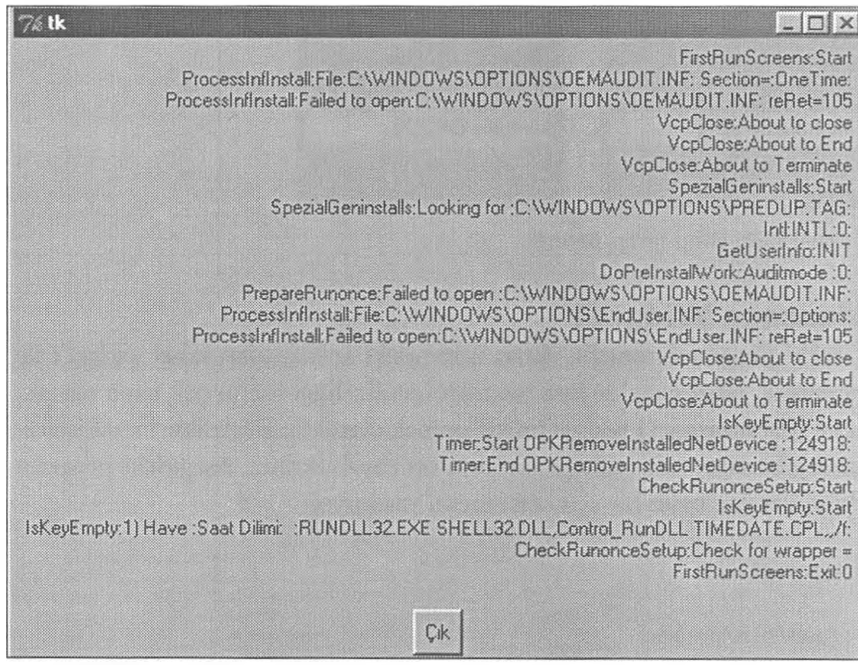
ana=Tk()

metin=open("c:\\frunlog.txt").read()
M=Message(text=metin)
M.config(justify="right")
M.pack()

D=Button(text="Çık")
```

```
D.config(command=sys.exit)
D.pack()
ana.mainloop()
```

Bu program çalıştırıldığında **c:\frunlog.txt** dosyası okunup bir pencere üzerine yazılacak ve altına bir çıkış düğmesi konulacaktır (**Şekil-16.23**). **justify** argümanının alabileceği değerler; sola yaslamak için **left** (ön tanımlı), ortalamak için **center** ve sağa yaslamak için ise **right**.



Şekil-16.23: Mesaj widget'i ile metin sola yaslanmış görünüyor.

Checkbutton Widget'i

Belirtilen öğenin seçilip seçilmesi için kullanılabilecek olan bu widget html bilenlere tanıdık gelecektir. Kullanımı ilk başlarda biraz zor gibi gelecektir, ancak diğer GUI araçlarını biliyorsanız, bunun ne kadar da kolay olduğunu göreceksiniz. Bu düğmelerin tıklanıp tıklanmadığını variable argümanının değerine bakarak anlayabilirsiniz. Bu argümana Tkinter sınıflarından olan **IntVar()** tipi bir nesne atarsanız, bu nesnenin değerini **get()** özelliği ile alabilirsiniz. Bu değer seçili düğmeler için1, diğerleri için 0 olacaktır. Herhangi bir düğmenin herhangi bir zamanda seçili durumunu değiştirmek için **set()** özelliğinden yararlanabilirsiniz.

Aşağıdaki program kullanıcıya hangi dilleri bildiğini soruyor. Python ön tanımlı olarak seçili geliyor. "Seçimler" düğmesine tıklandığında durumları ekrana yazılıyor.

program16-20

```
#!/usr/bin/python

import sys
from Tkinter import *

ana=Tk()

def yaz():
    print var1.get(), var2.get(), var3.get()

L=Label(text="Hangi Dilleri biliyorsunuz?")
L.pack()

TD1=Checkbutton(text="Perl")
var1=IntVar()
TD1.config(variable=var1)
TD1.pack(anchor=NW)

TD2=Checkbutton(text="Python")
var2=IntVar()
TD2.config(variable=var2)
TD2.pack(anchor=NW)
var2.set(1)

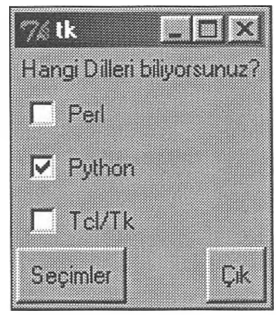
TD3=Checkbutton(text="Tcl/Tk")
var3=IntVar()
TD3.config(variable=var3)
TD3.pack(anchor=NW)
```

```
D1=Button(text="Seçimler")
D1.config(command=yaz)
D1.pack(side="left")

D2=Button(text="Çık")
D2.config(command=sys.exit)
D2.pack(side="right")

ana.mainloop()
```

Programı yazıp çalıştırdığımızda **Şekil-16.24**'teki gibi bir pencere ile karşılaşmanız gerekir.



Şekil-16.24:
Tıklama düğmeleri.

Şekil-16.24'teki pencerede seçimler düğmesine tıklandığında ekrana üç seçeneğin durumu bir satırda basılacaktır. Seçili olanlar 1, diğerleri sıfır şeklinde olacaktır.

Bu tür düğmeler kullanılırken genelde sözlükler tercih edilir ki, hem kod yazılımı kısalsın hem de değerleri öğrenmek kolay olsun. Şimdi yukarıdaki kodu Python programcısı gibi yazalım:

program16-21

```
#!/usr/bin/python

import sys
from Tkinter import *

ana=Tk()
```

```
ProgramlamaDilleri=("perl", "Python","Tcl/Tk")

def yaz():
    for p in ProgramlamaDilleri:
        print sonuc[p].get(),
    print
def temizle():
    for p in ProgramlamaDilleri:
        sonuc[p].set(0)

L=Label(text="Hangi Dilleri biliyorsunuz?")
L.pack()

sonuc={}

for p in ProgramlamaDilleri:
    sonuc[p]=IntVar()
    Checkbutton(text=p, variable=sonuc[p]).pack(anchor=NW)

D1=Button(text="Seçimler")
D1.config(command=yaz)
D1.pack(side="left")

D2=Button(text="Temizle")
D2.config(command=temizle)
D2.pack(side="right")

ana.mainloop()
```

Radiobutton Widget'i

Kullanıcıya seçim yapma imkanı veren bir diğer düğme şekli de radio düğmeleridir. Bu düğmeler ile genelde bir grup oluşturulur ve kullanıcıya bu gruptan bir tanesini seçebilme hakkı verilir. Her ne kadar Checkbutton'lar gibi birden çok seçim yapabilme hakkı verilebilse de genel eğilime ters düştüğünden kitabımızda anlatmayacağız.

Radiobutton'larda **variable** argümanı bir grup oluşturmak için kullanılır. Bu argümanın değeri aynı olan tüm düğmeler bir grup oluşturur ve sadece bir tanesinin seçilme hakkı olur. Buna ek olarak hangisinin seçildiğini anlamak üzere bir de **value** argümanı vardır. Checkbox'larda olduğu gibi seçim otomatik olarak bir IntVar'a atanmaz, bunun yerine düğmelere verilecek **command** argümanı ile bir değişkenin değeri, seçim yapılan düğmenin value argümanında bulunan değere atanabilir. Oluşturduğunuz grubun bir tanesini ön tanımlı olarak seçilmiş hale getirebilirsiniz. Bunun için, düğme nesnesinin **select()** özelliğini kullanabilirsiniz.

Aşağıdaki programda kullanıcıya ne içmek istediği soruluyor. Ön tanımlı olarak "Su" işaretli veriliyor. Kullanıcının seçimi icecek değişkenine atanıyor ve istenilen herhangi bir zamanda sonuç "Seçim" düğmesi ile ekrana yazdırılıyor.

program16-22

```
#!/usr/bin/python

import sys
from Tkinter import *

ana=Tk()

icecekler=("Çay", "Kahve","Su")

icecek='Su'

def yaz():
    print icecek

def degistir(x):
    global icecek
    icecek=x

L=Label(text="Ne içerdiniz?")
```

```
L.pack()

rb={}

for i in icecekler:

rb[i]=Radiobutton(text=i, variable="icecek", value=i,
                  command=(lambda i=i: degistir(i)))
    rb[i].pack(anchor=NW)

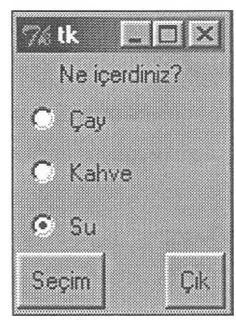
rb["Su"].select()

D1=Button(text="Seçim")
D1.config(command=yaz)
D1.pack(side="left")

D2=Button(text="Çık")
D2.config(command=sys.exit)
D2.pack(side="right")

ana.mainloop()
```

Programı çalıştırdığımızda **Şekil-16.25**'deki pencereye kavuşuyoruz.



Şekil-16.25: Tekli seçim düğmeleri

Hepsi Bir Arada

Yukarıda yapmış olduğumuz örnekleri bir araya toplayalım. Bir ekranda widget sayısı çoğaldıkça, neyi nereye nasıl yerleştireceğinizi önce kafanızda biraz canlandırmanız gerekiyor. Kullanıcıya bir anket penceresi hazırlayalım, bu ankette birbirinden bağımsız iki soru olsun: Birisi kullandığı diller,

diğeri yaşamak istediği ev tipi. İki tane düğme koyalım, birisi çıkmak için, diğeri kayıt için ve de adını soyadını yazabileceği bir metin giriş widget'i. Bu programı şöyle yazabiliriz:

program16-23

```
#!/usr/bin/python

from Tkinter import *

ana=Tk()

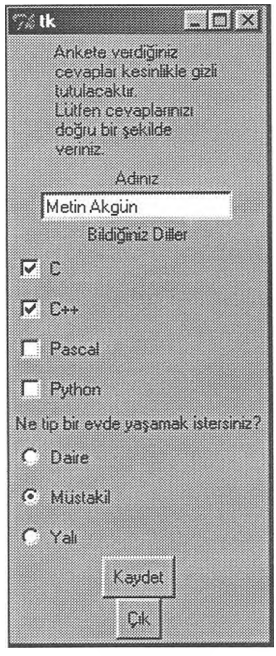
Message(text="""Ankete verdiğiniz cevaplar kesinlikle gizli
tutulacaktır.
Lütfen cevaplarınızı doğru bir şekilde veriniz.""").pack()
Label(text="Adınız").pack()
Entry().pack()

Label(text="Bildiğiniz Diller").pack()
for d in ["C","C++","Pascal","Python"]:
      Checkbutton(text=d).pack(anchor=NW)

Label(text="Ne tip bir evde yaşamak istersiniz?").pack()
for d in ["Daire","Müstakil","Yalı"]:
      Radiobutton(text=d, value=d,
variable="evtipi").pack(anchor=NW)

Button(text="Kaydet").pack()
Button(text="Çık", command=ana.quit).pack()
ana.mainloop()
```

Bu program çalıştırıldığında kullanıcıya hiç te sevimli gelmeyecek, her şeyin birbiri içine girdiği **Şekil-16.26**'deki gibi bir pencere ortaya çıkacaktır.



Şekil-16.26: Öğrendiğimiz widget'lerden bir çorbası.

Eğer, kullanıcıların bu ankete cevap vermesini istiyorsanız estetiğe dikkat etmelisiniz. Yaptığınız pencerelerin sanatsal değeri olmasa bile en azından çirkin görünmemelidir. Şimdi **Frame** widget'ini de kullanarak yukarıdaki programı biraz daha yüzüne bakılır hale getirelim.

program16-24

```
#!/usr/bin/python
from Tkinter import *

ana=Tk()
ana.title("Mini Anket")

M=Message(text="""Ankete verdiğiniz cevaplar kesinlikle gizli
tutulacaktır.
Lütfen cevaplarınızı doğru bir şekilde veriniz.""")
M.config(relief=GROOVE, width=250)
M.pack(expand=YES, fill=X)
```

```
F1=Frame(borderwidth=1,relief=RAISED)
Label(F1,text="Bildiğiniz Diller").pack()
for d in ["C","C++","Pascal","Python"]:
    Checkbutton(F1, text=d).pack(anchor=NW)
F1.pack(pady=4, side=LEFT)

F2=Frame(borderwidth=1,relief=RAISED)
Label(F2,text="Ne tip bir evde yaşamak istersiniz?").pack()
for d in ["Daire","Müstakil","Yalı"]:
    Radiobutton(F2, text=d, value=d,
variable="evtipi").pack(side=LEFT)
F2.pack(side=TOP, padx=4,pady=4)

F3=Frame()
Label(F3,text="Adınız").pack(side=LEFT)
Entry(F3).pack(side=RIGHT)
F3.pack(side=TOP, pady=8)

F4=Frame()
Button(F4,text="Kaydet", width=8).pack(padx=10,side=LEFT)
Button(F4,text="Çık", width=8,
command=ana.quit).pack(padx=10,side=RIGHT)
F4.pack(side=TOP, pady=8)

ana.mainloop()
```



Şekil-16.27: Widget'lar düzenli bir şekilde yerleştirilmeli.

Elbette bundan daha iyisi yapılabilir. En azından göze çok kötü görünmüyor: Şekil-16.27.

Diğer Geometri Yöneticileri

Şimdiye kadar widget'ları ekranda göstermek için **pack()** metodunu kullandık. Widget'ların yerlerini belirtmek içinse **side**, **anchor** gibi argümanlar ve **frame** kullandık. Tüm bunları küçük programlarda yapmak kolay, fakat program büyüdükçe ve ekranda gösterilecek widget sayısı çoğaldıkça hem kod uzuyor hem de, nereye nasıl yapıştıracağımızı kestirmek zorlaşıyor.

Grid

Yukarıdaki zorlukları aşmak üzere, Tkinter'da bir geometri yöneticisi olan **grid**'i kullanabilirsiniz. Böylelikle, ana pencereniz bir tablo gibi olacak ve siz bu tablonun istediğiniz hücresine istediğiniz widget'i koyabileceksiniz. Grid'lerle birlikte frame'ler de sıkça kullanılmaktadır.

Önceki örneklerimizde kullandığımız öğrenci not sisteminde, not girmek için hazırlanan ekranı yapmaya çalışalım:

program16-25

```
#!/usr/bin/python

from Tkinter import *
ana=Tk()

T=["Öğrenci No","Adı Soyadı","Matematik","Fen Bilgisi",
"Sosyal Bilgiler"]

c=0
for i in T:
    Label(text=i).grid(row=c, column=0)
    Entry().grid(row=c, column=1)
    c=c+1

ana.mainloop()
```

Gördüğünüz gibi hiç **pack()** kullanmadık. Program çalıştırıldığında **Şekil-16.28**'deki ekran oratava çıkacaktır.



Şekil-16.28: Grid kullanılarak widget'ların pencere üzerine yerleştirilmesi

Bu programda widget'lar pencere üzerine yerleştirilirken, pencere adeta bir tablo gibi oldu ve **row** (satır), **column** (sütun) numaraları **grid()** fonksiyonuna verilerek, widget'in nereye yerleştirileceğine karar verildi.

Place

Diğer bir pencere yöneticisi ise **place**'dir. Bu yönetici ile, istediğiniz widget'i, pencerenizin istediğiniz noktasına, istediğiniz büyüklükte yerleştirebilirsiniz. Kullanması biraz sıkıcı olabilir, ancak oldukça estetik tasarımlar yapabilirsiniz. Küçük bir örnek vererek bitirelim:

program16-26

```
#!/usr/bin/python

from Tkinter import *

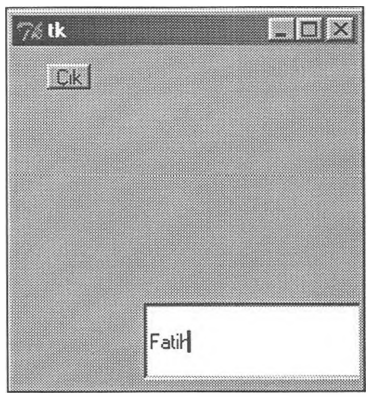
ana=Tk()

B=Button(text="Çık")
B.place(height=15, width=25, x=20, y=10)

E=Entry()
E.place(height=45, width=150, x=75, y=150)

ana.mainloop()
```

Yukarıdaki programda tüm değerlerin piksel cinsinden olduğunu unutmayın. Bu program çalıştığında **Şekil-16.29**'daki gibi bir pencere veriyor.



Şekil-16.29:
Place geometri yöneticisi ile widget'leri istediğiniz gibi yerleştirebilirsiniz.

Resimler

Tkinter ile çok çeşitli biçimdeki resimleri okuyup pencerenizin istediğiniz bölümüne yapıştırabilirsiniz. Resimleri okuma işlemi **PhotoImage()** fonksiyonu ile yapılabilir. Birçok widget, üzerinde resim göstermeye izin vermektedir.

Aşağıdaki programda etiket üzerine (metin yerine) **tcl.gif** resmi yerleştiriliyor. Çık düğmesinin üzerine bir metin değil **exit.gif** resmi yerleştiriliyor:

program16-27

```
#!/usr/bin/python

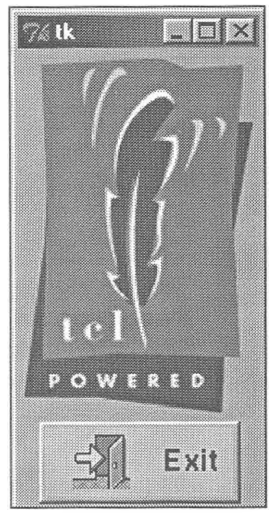
import sys
from Tkinter import *

ana=Tk()

tcl=PhotoImage(file="tcl.gif")
Label(image=tcl).pack()
cik=PhotoImage(file="exit.gif")
Button(image=cik, command=sys.exit).pack()

ana.mainloop()
```

Bu program çalıştırıldığında, Şekil-16.30'daki pencereyi görmeniz gerekiyor.



Şekil-16.30: Tkinter ile resimlerin çeşitli widget'ler üzerine yerleştirilmesi

Listbox ve Kaydırma Çubuğu

Bir listeden seçim yapılacaksa en iyisi bunu listbox'larla yapmaktır. Listbox'ların eleman sayısı genelde çok olduğu için, kenarına bir de kaydırnma çubuğu koymak iyi olacaktır. Aşağıdaki programda bu iki widget'i nasıl kullanacağınızı görüyorsunuz:

program16-28

```
#!/usr/bin/python

import os
from Tkinter import *

def secimiYaz(event):
    print L.curselection()

ana=Tk()

L=Listbox()
```

```
dosyalar=os.listdir("C:/Windows")

dosyalar.sort()

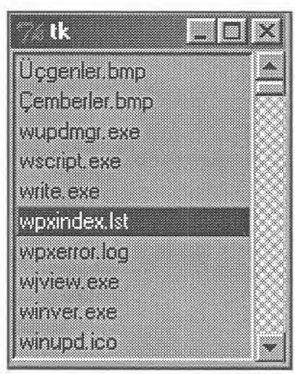
for i in dosyalar:
    L.insert(0, str(i))

sbar=Scrollbar()

sbar.config(command=L.yview)
L.config(yscrollcommand=sbar.set)
L.pack(side=LEFT)
L.bind("<Double-1>",secimiYaz)
sbar.pack(side=RIGHT, fill=Y)

ana.mainloop()
```

Bu program çalıştırıldığında **c:\windows** kalsöründeki tüm dosya ve klasörlerin bir listesini ekranda gösterecektir. Sol tarafındaki kyadırma çubuğu ile listenin alt kısımlarını görebilirsiniz. Herhangi bir elemana çift tıklarsanız, tıkladığınız elemanı ekranda göreceksiniz, böylelikle seçiminde nasıl yapılabileceğini anlatmış olduk. Örnek pencere **Şekil-1.31**'de



Şekil-16.31: Listbox ve kaydırma çubuğu widget'leri

Text Widget'i

Kullanıcıdan bilgi girişi yapmasını istediğimizde entry widget'ini kullanabilirsiniz, ancak uzunca bir metin girişi gerekiyorsa o zaman **Text** widget'ine ihtiyacınız vardır. Aşağıdaki örnekte basit bir metin girişi görüyorsunuz:

program16-29

```
#!/usr/bin/python

from Tkinter import *

ana=Tk()

T=Text(width=25, height=15)
T.pack()

ana.mainloop()
```

Programı çalıştırdığında kullanıcının karşısına metin girebileceği **Şekil-16.32**'deki gibi bir ekran çıkacaktır.



Şekil-16.32:
Metin giriş ekranı

Yukarıdaki programı biraz daha etkin bir şekle sokalım. Programımıza şu eklentileri yapalım:

- Kaydırma çubuğu
- Yazdıklarını kaydetme düğmesi
- Çıkış düğmesi

Kaydırma çubuğu için Scrollbar widget'ini kullanabiliriz. Metin alanına girilen bilgiler ise **get()** metodu ile alınabilir. Bu eklentileri aşağıdaki programda görebilirsiniz:

program16-30

```
#!/usr/bin/python

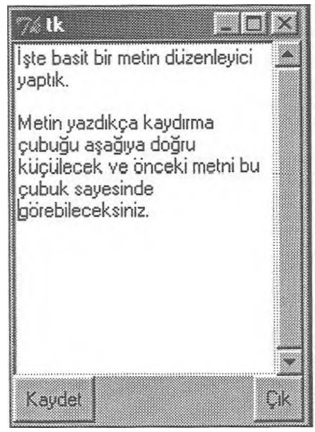
import sys
from tkFileDialog import *
from Tkinter import *
def kaydet():
    file=asksaveasfilename()
    if file:
        dosya=open(file,'w')
        dosya.write(T.get('1.0',END))
        dosya.close()

ana=Tk()
TF=Frame()
T=Text(TF, width=25, height=15)
T.pack(side=LEFT)
sbar=Scrollbar(TF)
sbar.config(command=T.yview)
T.config(yscrollcommand=sbar.set)
sbar.pack(side=RIGHT, fill=Y)
TF.pack()

Button(text="Kaydet", command=kaydet).pack(side=LEFT)
Button(text="Çık", command=sys.exit).pack(side=RIGHT)

ana.mainloop()
```

Yazdığımız bu programı çalıştırdığınızda **Şekil-16.33**'deki gibi oldukça basit bir metin düzenleyici ile karşılaşacaksınız.



Şekil-16.33: Python ile yazılmış basit bir metin düzenleyici.

OptionMenu Widget'i

Listbox'lar gibi bir seçim listesi sunar ve kullanıcının bir tanesini seçmesine izin verir. Web sayfalarındaki <select> girdisine çok benzemektedir, ancak görünüm her platformda UN*X tipindedir. Tix kullanarak windows benzeri combo box'lar oluşturabilirsiniz.. Aşağıdaki program kullanıcının ulaşım aracını soruyor, bir düğme sayesinde yaptığı seçim ekrana yazılıyor.

program16-31

```
#!/usr/bin/python

from Tkinter import *

ana=Tk()

def secimYaz():
    print degisken.get()

Fl=Frame()

Label(Fl,text="Ulaşım aracınız: ").pack(side=LEFT)
```

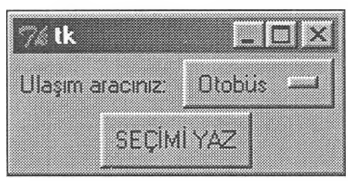
```
degisken=StringVar()
opt=OptionMenu(F1, degisken, "Dolmuş", "Otobüs", "Taksi",
"Kendi Arabam")
degisken.set("Otobüs")
opt.pack(side=RIGHT)

F1.pack()

Button(text="SEÇİMİ YAZ", command=secimYaz).pack()

ana.mainloop()
```

OptionMenu widget'inin ilk aldığı argüman, hangi pencerenin (ya da çerçevenin) çocuğu (üyesi) olduğudur. Bu programda **F1** çerçevesinin çocuğu olmuştur. İkinci aldığı argüman ise, bir cümle nesnesidir ve bu nesne **StringVar()** ile yapılır. **OptionMenu**'nun ilk değerinin atanması ve sonraki seçimlerin alınması için bu nesnenin **set()** ve **get()** metotları (fonksiyonları) sıra ile kullanılır. **Şekil-16.34**'de bu programın çalıştırıldığında oluşturduğu pencereyi görüyorsunuz.



Şekil-16.34: OptionMenu Widgeti

Menüler

Bir GUI'nin vazgeçilmez elemanı menülerdir. Menüler sayesinde kullanıcı programa çeşitli komutları gönderir ve yapacaklarını bu şekilde belirtir. Tkinter çok zengin menü seçenekleri sunar, ancak biz burada en çok kullanılan, yalın halini anlatacağız. Daha ileri düzey menü oluşturmak için ileri Python kitaplarına bakabilirsiniz.

Menüler oluştururken şu iki madde çok önemlidir:

Ayırım Çizgileri

Bu çizgiler sayesinde menü elemanları birbirinden ayrılarak gruplama yapılabilir. Bu **add_seperator** ile yapılır.

Klavye Bağlantıları

Her ne kadar kullanıcıya kolaylık olsun diye GUI geliştirilmiş ise de hızlı bilgisayar kullanabilmek için ille de klavye kullanmak gerekiyor. Bir elin klavyede bir elin farede olması verimliliği çok düşürmektedir. Verimliliği artırmak için hemen her fonksiyon için klavye tuş kombinasyonları kullanılarak kullanıcının mümkün olduğu kadar fareden uzak kalması sağlanır. Örneğin birçok programda **Ctrl+B** tuş takımı yazımın veya seçimin kalın hale gelmesini sağlar. Bunu araç çubuğundan yapmak için en az beş saniyeniz gider. Tkinter'de widget'lara **underline** argümanı verilerek yapılır.
Şimdi basit bir menü yapalım:

program16-32

```
#!/usr/bin/python

from Tkinter import *

def yeni():
    print "Yeni bir dosya açılacak"
def ac():
    print "Eski bir dosya açılacak"

def yazdir():
    print "Yazıcıya gönderilecek"

ana=Tk()

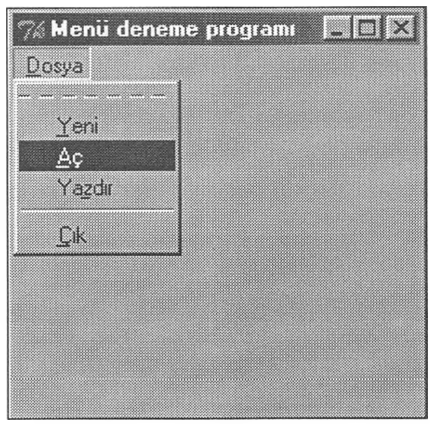
top=Menu(ana)
ana.title("Menü deneme programı")
ana.config(menu=top)

dosya=Menu(top)
dosya.add_command(label="Yeni", underline=0, command=yeni)
dosya.add_command(label="Aç", underline=0, command=ac)
dosya.add_command(label="Yazdır", underline=2, command=yazdir)
```

```
dosya.add_separator()
dosya.add_command(label="Çık", underline=0, command=ana.quit)
top.add_cascade(label="Dosya", underline=0, menu=dosya)

ana.mainloop()
```

Bu programı çalıştırırsanız **Şekil-16.35**'deki gibi bir pencereye sahip olursunuz.



Şekil-16.35:
Tkinter menüleri

Şimdi bu programda önemli gördüğümüz birkaç noktayı açıklayalım:

- Bir menü oluşturmak için **Menu()** fonksiyonu (aslında bir widget) kullanılır. Bu menünün alacağı ilk argüman kimin çocuğu olacağıdır. Eğer hiçbir şey yazmazsanız ana pencereye ait bir menü olacaktır. Bir pencereye menü ekleyebilmek için mutlaka ana pencereye ait bir menü oluşturulmalıdır. Yukarıdaki programda bunu **top=Menu(ana)** satırı ile yaptık.
- Eklenecek alt menüler ana menü cinsinden birer menüdürler. Programımızda "Dosya" menüsünü oluşturmak için, top menüsünün bir çocuğu olduğunu belirtecek şekilde **dosya=Menu(top)** satırı ile yapmış olduk. Bu sayede yeni bir menü tanımlandı. Artık bu menüye çeşitli komutlar ekleyebiliriz. Eğer **Menu()** fonksiyonuna **tearoff=0** parametresini vermezseniz, menüde ilk olarak görünecek kesikli çizgilerden oluşan bir komut oluşturulur. Bu kesikli çizgileri seçen bir kullanıcı, o menüyü ayrı bir üstdüzey penceresi yapmış olur. Yukarıdaki programı

çalıştırdıktan sonra, "Dosya" menüsünü açıp ve kesikli çizgileri sanki bir menüymüş gibi seçerek etkiyi görünüz.

- Bir menüye komut eklemek için o menü nesnesine ait **add_command()** fonksiyonu kullanılır. Bu fonksiyonun alacağı parametreler şöyledir:
  label: komutun ekranda görünen metnini yazdırır.

  underline: Kısayol tuş takımının metnin ilk karakter sıfır olmak üzere kaçıncı karakteri olduğunu belirtir.

  command: Menü seçildiğinde hangi fonksiyonun çağrılacağını belirtir.
- Menüleri birbirinden ayırmak için kullanacağınız çizgi, menü nesnesinin **add_seperator()** fonksiyonu ile gerçekleştirilir.
- Menünün gerçek anlamda bağlanması, bir üst menü nesnesinin **add_cascade()** fonksiyonu ile yapılır. Yukarıdaki programda bir üst menü nesnesi **top**'dır. Bu fonksiyonun alacağı parametreler **add_command** ile aynıdır.

Menülerle ilgi yaptığımız bu açıklamalardan sonra bir de altmenülerin nasıl oluşturulduğuna bakalım:

program16-33

```
#!/usr/bin/python

from Tkinter import *

def yeni():
    print "Yeni bir dosya açılacak"
def ac():
    print "Eski bir dosya açılacak"

def yazdir():
    print "Yazıcıya gönderilecek"

ana=Tk()
```

```
top=Menu(ana)
ana.title("Menü deneme programı")
ana.config(menu=top)

dosya=Menu(top, tearoff=0)
dosya.add_command(label="Yeni", underline=0, command=yeni)
dosya.add_command(label="Aç", underline=0, command=ac)
dosya.add_command(label="Yazdır", underline=2, command=yazdir)
dosya.add_separator()
dosya.add_command(label="Çık", underline=0, command=ana.quit)
top.add_cascade(label="Dosya", underline=0, menu=dosya)

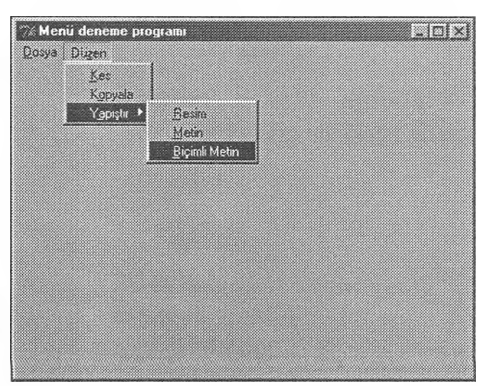
duzen=Menu(top, tearoff=0)
duzen.add_command(label="Kes", underline=0)
duzen.add_command(label="Kopyala", underline=1)
top.add_cascade(label="Düzen", underline=2, menu=duzen)

duzenYapistir=Menu(duzen, tearoff=0)
duzenYapistir.add_command(label="Resim", underline=0)
duzenYapistir.add_command(label="Metin", underline=0)
duzenYapistir.add_command(label="Biçimli Metin", underline=0)

duzen.add_cascade(label="Yapıştır", underline=1,
menu=duzenYapistir)

ana.mainloop()
```

Bu program çalıştığında **Şekil-16.36**'daki pencereyi verecektir.



Şekil-16.36:
Tkinter alt menüleri

Elbette istediğinizde menülere ikonlar koyabilirsiniz. Bunun için anlattığımız resimler konusuna bakmanız yeterli.

Çizimler

Tkinter'da çizim yapabileceğiniz widget **canvas** alanıdır. Bu alan içerisine çeşitli çizimler yapabilirsiniz. Burada amacımız hepsini öğretmek değil, sadece neler yapılabileceğine dair temel şekiller ve örnekler vermektir. Daha detaylı örnek ve bilgileri internetten bulabilirsiniz.

program16-33

```
#!/usr/bin/python

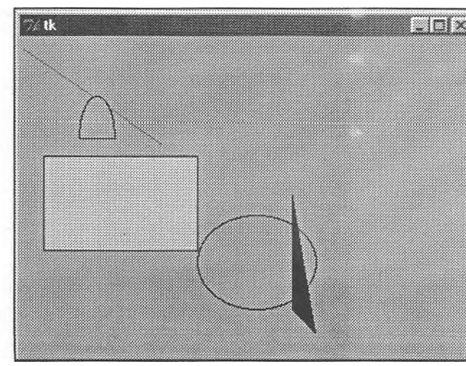
from Tkinter import *
from Canvas import *

ana=Tk()

cv=Canvas()
Line(cv, 3, 10, 120,90, fill="red")
Rectangle(cv, 20, 100, 150, 180, fill="green")
Arc(cv, 50,50,80,120, extent=180, fill="#ccbbdd")
Oval(cv, 150,150,250,230)
Polygon(cv, 230,230, 250,250,230,130)
cv.pack()

ana.mainloop()
```

Programın verdiği görüntü **Şekil-16.37**'de görülüyor.

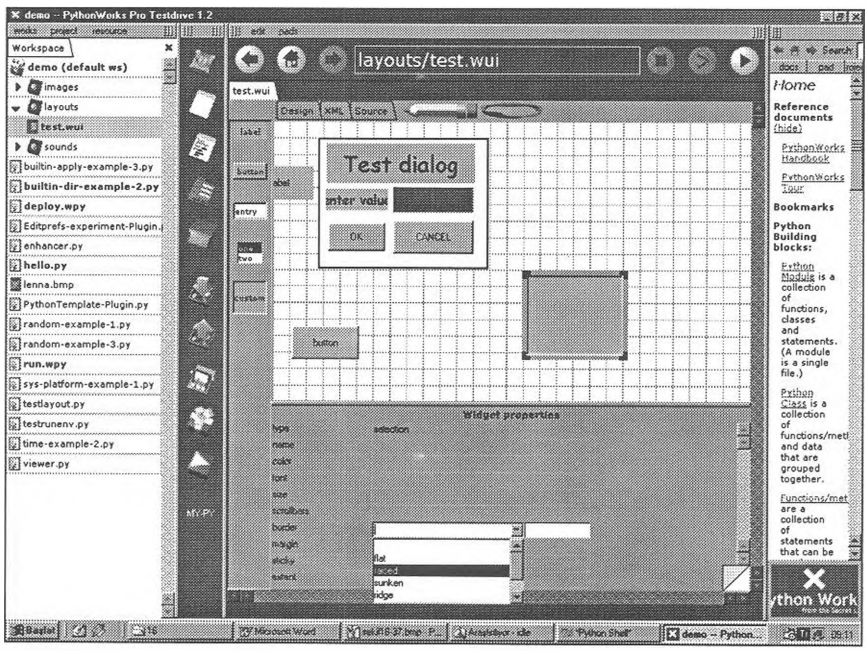


Şekil-16.37: Tkinter ile çizim yapabilirsiniz.

Tkinter ile yapabilecekleriniz burada anlattıklarımızla sınırlı değil elbette. Bu araç ile istediğiniz her şeyi yapabilirsiniz. Internette çok değişik program örnekleri bulmanız mümkün. Tkinter kullanılarak Wordpad benzeri bir program bile yazılmıştır, hatta grail web tarayıcıda python+tkinter ile yazılan en ciddi programlardır.

Görsel Araçlar

Python ile GUI programları geliştirmek istiyorsanız size birkaç görsel araç önerebiliriz. Bunlardan birisi Microsoft.NET için hazırlanmış VisualPython'dur. Bunun *http://www.activestate.com* adresinden indirebilirsiniz. Diğer bir araç ise Secret Labs tarafından geliştirilmiş, geliştiricileri arasında Python'un büyük isimlerinden Frederik Lundh'unda bulunduğu, çoğu kısımların yine Python ile yazıldığı PythonWorks Pro'dur. Bu daha önce alıştığınız program geliştirme ortamlarına çok benzemekte, çevirim içi yardımları, olağanüstü Python metin düzenleyicisi, görsel GUI geliştirme aracı ile, Python alanında yapmak istediğiniz her şeyin bir araya getirilmesinden oluşan mükemmel bir araçtır. **Şekil-16.38**'de örnek bir penceresini gördüğünüz aracı incelemek için deneme sürümünü *http://www.pythonware.com* adresinden indirebilirsiniz. PythonWorks hem Windows hem de Linux ortamında çalışabiliyor.



Şekil-16.38: PythonWorks olağanüstü bir RAD'dı.

Web Programlama



Bu bölümde Python ile dinamik web sayfalarının nasıl yapılacağını öğreneceğiz. Değişik platformlarda ne tür web programcılığı yapabileceğinizi kısa kısa anlatacağız. Web programcılığının tüm yönlerini anlatmak belki başka bir kitabın konusu olacak.

Web Programcılığı Nedir?

Web programcılığını klasik programcılıktan ayıran birçok özellik vardır:

1. Öncelikle program çok kısa sürede yazılmalı ve mümkün olan en az hata ile çalışmalıdır. Çünkü web uygulamaları çok çabuk değişmekte, gelişen teknoloji ve kullanıcıların istekleri doğrultusunda ya başka bir uygulamaya geçilmekte ya da değiştirilmektedir. Dolayısı ile çok çabuk yazılmalı ve hemen uygulamaya konulmalıdır.
2. Yazdığınız program oldukça hızlı çalışmalıdır. Program bir kişi değil bir kullanıcı grubuna hitap etmektedir.
3. Program bilinmedik kişiler tarafından çalıştırılacağından güvenli olmalıdır.
4. Her platformda mümkün olan en az değişiklikle çalışabilmelidir. Web sunucularından bağımsız bir dilde yazılmalıdır. IIS için yazdığınız kod, Apache'de de çalışabilmelidir.
5. Yazdığınız program günün birinde mutlaka başkaları tarafından yeniden düzenleneceğinden açık kaynak kodlu olmalı ve düzeltilmesi kolay olmalıdır.
6. Kullanacağınız dil size hiçbir sınır koymamakla birlikte, web programcılığı için gerekli olan tüm araçları sunmalıdır.

Bunlar şu anda bizim tarafımızdan öngörülen özelliklerdir. Bunların sayısı ve niteliği çeşitli web programcıları tarafından artırılabilir. Tüm bu özellik-

lerin çoğuna sahip programlar yazmak elbette tüm dillerle mümkündür. Hiç kimse size C++ ile web uygulaması yazmamanızı öğütlemez, ancak yukarıdaki kıstasları göz önüne aldığınızda çok da akıllıca bir iş yapmadığınız ortaya çıkacaktır.

Web programcılığı son yılların en çok rağbet gören alanlarından birisidir. Bilgisayar üzerindeki uygulamaların çoğu artık webe taşınmıştır. Bankacılık işlemlerinden tutun, alışverişe, oyun oynamaya kadar çok geniş bir topluluğa hitap etmektedir. Her ne kadar ülkemizde henüz internetçilikten para kazanılmasa da (uygulanan yanlış politikalardan dolayı), diğer ülkelerde artık ticaret webe kaymaktadır. Bir mağaza açmanın internet maliyeti ile fiziksel maliyeti karşılaştırılamaz. Öte yandan fiziksel mağazadaki müşteri sayısı da internetteki ile kıyaslanamaz. Ancak insanların rahat ve kolay yaşama arzuları tembellikle birleşince internetten alışveriş yapma eğilimi de gitgide artmaktadır. İşte insanlara hizmet verecek olan bu programların hepsini biz web programcılığı altında topluyoruz. O halde, web programı yazabilmek artık tüm programcıların asli görevlerinden birisi olmuştur. Bunlarla birlikte programcılar, bu programları nasıl ve hangi dille yazacaklarına karar verirken yukarıda saydığımız özelliklere de dikkat etmek zorundalar.

Tüm bunların doğal bir sonucu olarak, sadece web programcılığı yapmak üzere geliştirilen birçok programlama dili mevcuttur. Bunlardan birisi PHP'dir. PHP ilk başlarda sadece dinamik web sayfaları oluşturmak üzere geliştirilmiş bir dildi. Öğrenmesi oldukça kolaydır, fakat uygulamaya geldiğinde, programcının çektiği eziyet tarifsizdir. Perl de bu amaçla geliştirilen dillerden birisidir.

Bir Microsoft teknolojisi olan ASP (**Active Server Pages**) bir programlama dili değildir, bir teknolojidir. Bu teknolojideki ana hedef web programcılığını tamamen kolaylaştırmaktır. Microsoft tarafından öngörülen dil VBScript'dir. Visual Basic nesnelerini kullanarak web sayfası yazma yöntemidir. Visual Basic programcıları, web programı yazmak için dünyanın emeğini ve vaktini harcarken, aynı işleri daha kolay ve daha hızlı yazan, güvenli ve performanslı bir dile ihtiyaç vardır. Bu dil kesinlikle size bir sınır koymamalı, web programcılığı için geliştirilmiş diye sizi bir yerde sıkıştırıp bırakmamalıdır.

Web programcılığı, teknoloji açısından iki gruba ayrılabilir. Bunlardan ilki ve hala çok tercih edileni CGI (Common Gateway Interface)'dir. Bu tür programcılıkta, program sanki metin tabanlı bir uygulama gibi yazılır ve çıktılar ekrana (standart çıktıya) yazılır. Çıktı html kodlarından oluşur. Kullanıcıdan gelen bilgiler ise ya çevre değişkenlerinden ya da standart girdiden alınır. Performansın düşük olması yavaş yavaş CGI'den kaçışa yol açmaktadır. Diğer bir yöntem ise, program kodlarının html sayfası içerisine yazılması ve bu kodların web sunucusunda hazır bulunan bir yorumcu tarafından yorumlanmasıdır. Bu tip programcılık hem hızlı hem de yazarlar için oldukça kolaylık sağlamaktadır. Bir diğer yararı ise web tasarımcılarının bu sayfaları kolayca düzenleyebilmesidir. PHP bu tip çalışan dillerden birisidir.

ASP temelde sunucunun bir sayfayı yayınlamadan önce içeriğinin bir yorumcu ile yorumlanması teknolojisidir. PHP ile benzer bir yöntem kullanılır, ancak ASP'de kullanılacak olan dil belli değildir. Sunucuya asp sayfasının kimin tarafından yorumlanacağını bildirirsiniz, IIS'de onu yorumlatır. Bu amaçla VBScript, Perl, Tcl ya da Python kullanabilirsiniz. Kitabımızda web programcılığının büyük bir kısmı ASP tarzında anlatılacaktır. ASP'yi Python ile yazdığınızda bu PSP (**Python Server Pages**) olarak adlandırılır. PSP hem UN*X hem de Windows platfromlarında çalışabilir.

İlk olarak CGI programcılığı üzerinde duracağız. Kitabımızı yazarken Linux ve Windows platformlarını kullandık. Web sunucusu olarak da Linux'ta Apache, Windows'da hem Apache hem de Personal Web Server (IIS4) kullandık. Her sunucu için neler yapılması gerektiğini anlattık.

Neler Bilmeliyim?

Bu bölümden yeteri kadar yararlanabilmek için:

- Basit html kodlamayı
- Form elemanlarını ve kullanımlarını
- Bir html sayfasının yapısını
- Web sunucusu ve web tarayıcısı kavramlarını
- Linux için: Linux kullanmayı
- Windows için: IIS kullanmayı
- Python Programlama Dilini bilmeniz gerekmektedir.

CGI Programlama

En yaygın web programcılığı metodu olduğunu söylemiştik. Her programlama dili ile yapılabilir. Python bu iş için sanki biçilmiş kaftan gibidir. Cgi modülü sayesinde kullanıcıdan gelen her türlü veri en iyi şekilde parse edilerek kullanıma hazır hale getirilir. Yazılan kod tüm platformlarda çalışabilir. Güvenli bir web sitesi yapmak istiyorsanız bu işi mutlaka CGI ile yapmalısınız. Çünkü tüm kontrol sizin elinizin altındadır. Kullanıcıdan gelen bilgiler direk sizin programınıza hiçbir değişikliğe uğratılmadan verilir. Web sunucusunda hazır olarak bir şeyin bulunmaması sayesinde yapılan saldırılar amacına ulaşamaz.

Kod yazılımı CGI programcılığında oldukça kolay ve anlaşılırdır. Progamın standart çıktısı sunucu tarafından hiçbir değişikliğe uğratılmadan, doğrudan istemcinin web tarayıcısına gönderilir. Böylelikle kod denetleme de oldukça kolaylaşmış olur.

Yazdığınız programın görsel tasarımı bir web tasarımcısı tarafından yapılacaksa iş biraz zorlaşmaktadır. Ancak bu tür programlar yazıyorsanız html template'leri kullanarak işinizi daha profesyonel bir şekilde yapmış olursunuz, tasarımcı da programınızı hiçbir zaman görmez ve Python kodları arasında boğulup kalmaz.

CGI programcılığında genel eğilim her işin bir ana program tarafından ele alınmasıdır. ASP ve diğer yazılımlarda (PHP gibi) kullanıcı ile etkileşim genelde farklı programlar (ya da sayfalar) aracılığı ile yapılır.

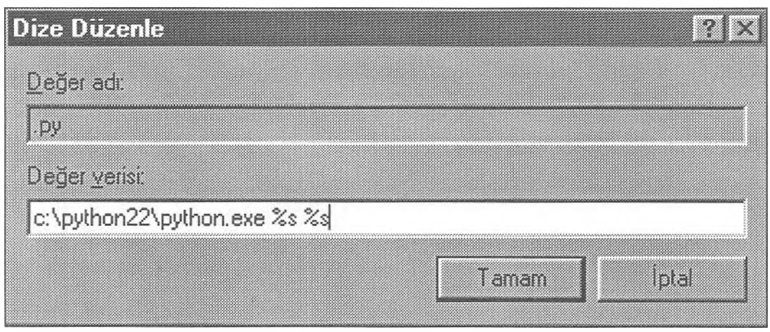
CGI Performansın düşük olması nedeniyle yavaş yavaş terkedilmektedir. Her ne kadar FastCGI metot ve modülleri geliştirilse de istenilen verimliliğe ulaşamamaktadır. Ancak hala en güçlü ve güvenilir programlar CGI ile yazılmaktadır.

UN*X tipi işletim sistemi kullanıyorsanız, Python ile CGI programları yazmak için sunucuda herhangi bir ayar yapmanız gerekmiyor. Tek aklınızda tutmanız gereken Python betiğini yazdıktan sonra onu **chmod** komutu ile çalıştırılabilir yapmaktır. Windows kullanıcılarının ise ister Personal Web Server (Kişisel Web Sunucusu) kullansın isterse IIS'nin herhangi bir versiyonunu kullansın registery ayarları yapması gerekiyor. Sunucunun **.py** uzan-

tılı programları kime yorumlatacağını registery ayarları yapılarak belirtilebilir. Şimdi "Başlat / Çalıştır" yolunu izleyin ve kutucuğa "regedit" yazın. Açılan pencerede aşağıdaki yolu izleyin.

HKEY_LOCAL_MACHINE \ System \ CurrentControlSet \ Services \ W3SVC \ Parameters \ Script Map

Yolunu takip ettikten sonra "Düzen / Yeni / Dize Değeri" menüsüne gelin. Kayıt düzenleyicisinin sağ tarafındaki pencerede "Yeni Değer #1" şeklinde seçili bir bölge bulacaksınız. Burdaki metni silerek içeriğini ".py" yapın. Daha sonra üzerine çift tıklayın ve açılan penceredeki yazılabilir kutucuğa **Şekil-17.1**'deki gibi "**c:\python22\python.exe %s %s**" değerini girin. "Tamam" düğmesine tıkladıktan sonra kayıt aracını kapatın ve değişikliğin etkin olması için makinanızı kapatıp açın. Eğer Python yorumlayıcısı başka bir patikada bulunuyorsa burada gerekli değişikliği yapın.



Şekil-17.1: IIS'de Python script'lerinin kayıt dosyasına eklenmesi.

IIS'de CGI scriptleri (betikleri) eğer bir değişiklik yapmamış iseniz ön tanımlı olarak **C:\Inetpub\scripts** klasörüne konulur ve tarayıcıdan istenirken de adres satırına **http://<makineAdı>/scripts/<scriptAdı>** yazılır.

Windows üzerinde Apache kullanıyorsanız ve Python ile CGI programcılığı yapacaksanız, herhangi bir şey yapmanıza gerek yok. Sadece yazacağınız programı **cgi-bin** klasörüne koymalısınız ve programın ilk satırı tıpkı UN*X'te olduğu gibi Python yorumlayıcısının patikası olmalıdır. Eğer Windows/Apache kullanıyorsanız aşağıdaki tüm programlarda ilk satırda bulunan "#!/usr/bin/python" yerine "C:/Python22/Python.exe" yazmanız gerekmektedir. Python patikanız farklı ise gerekli değişikliği kendinize göre

yapın. Apache'de CGI'ler hangi platformda olursa olsun tarayıcıdan **http://<makinaAdı>/cgi-bin/<scriptAdı>** şeklinde istenir.

Merhaba CGI

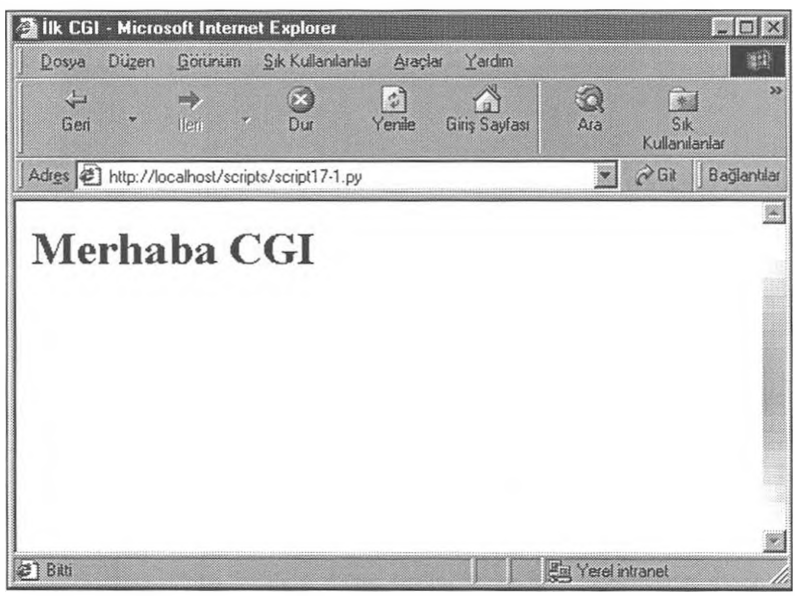
Yukarıdaki işlemleri tamamlamışsanız, gelin ilk CGI programımızı yazalım.

script17-1.py

```
#!/usr/bin/python

print "<html>"
print "<head><title>İlk CGI</title></head>"
print "<body>"
print "<h1>Merhaba CGI</h1>"
print "</body>"
print "</html>"
```

Bu kodları gereken yere **script17-1.py** adıyla kaydettikten sonra hemen tarayıcınızı açın ve adres satırına **http://localhost/scripts/script17-1.py** (Apache kullanıyorsanız **http://localhost/cgi-bin/script17-1.py**) yazın. Web tarayıcınız **Şekil-17.2**'deki gibi olacaktır.



Şekil-17.2: İlk CGI programınız

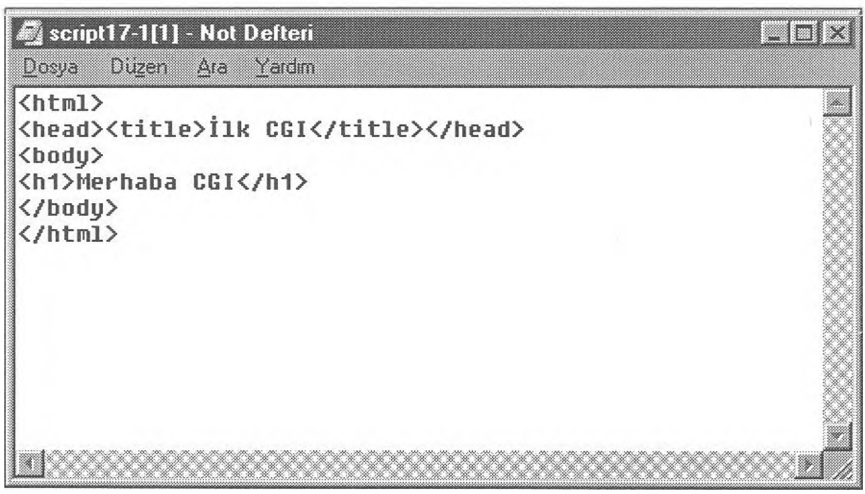
Şimdi bu programda neler olup bittiğine bakalım. Birinci satırın anlamını biliyor olmalısınız. İkinci satırda bulunan

```
print "Content-Type: text/html"
```

CGI protokolü gereği bir CGI programında bulunması gereken ilk satırdır. (Elbette daha başka başlıklar gönderebilir. Örneğin redirect gibi). Bu satır sayesinde geri kalan çıktının bir html sayfası olacağı ve bunun istemciye bildirileceği anlaşılır. Eğer bir resim gönderilecekse o zaman text/html yerine image/<resimBiçimi> yazılır, örn: image/gif gibi.

Başlık yazıldıktan sonra mutlaka bir satır atlanmalıdır. Bu da sonraki print komutu ile yapılıyor. Aslında başlığı yazarken sonuna "\n" koymanız da yeterlidir.

Bundan sonrakiler ise sanki ekrana bir html kodu yazıyormuş gibidir. Tüm bu işlemler sunucuda yapılır ve istemciye sadece standart çıktıya yazılan kod gönderilir. **Şekil-17.3**'de bu sayfanın kaynak kodu görünüyor (Tarayıcı üzerinde farenin sağ tuşuna tıklayın ve "Kaynağı Görüntüle"yi seçin)



Şekil-17.3: İlk CGI programınızın tarayıcıdaki kaynak kodu.

Yukarıda gördüğünüz gibi yaptığımız işlem basit bir html sayfasının kodunu üretmektir.

CGI Modülü

Birçok dinamik web sayfası kullanıcıdan birtakım bilgiler alır ve oluşturulacak sayfa buna göre biçimlenir. Kullanıcıdan veri almanın iki yolu vardır. Bunlardan biri GET, diğeri POST'dur. İlk yöntemde kullanıcı bilgileri, programın çalıştığı ortama çevre değişkeni ile aktarılır. Diğerinde ise programa standart girdiden verilir. Ancak bu verileri alıp kullanıcının gönderdiği form isimlerindeki değerlere ayrıştırmak oldukça yorucu ve gereksiz bir iştir. Bu işlemler hemen her programlama dilinde bulunan bir CGI modülü ile yapılır. Python olağanüstü bir **cgi** modülüne sahiptir. Bu modül kullanıcıdan gelen veri ister GET, isterse POST ile olsun programcıyı hiç ilgilendirmez (Not: Aynı anda bunlardan sadece birisi ile veri gelebilir). Kullanıcıdan gelen veriler sanki bir sözlükmüş gibi kullanılabilir. Şimdi bu modülü kullanarak, kullnıcıdan GET ile gelen ilk veriyi alalım:

script17-2.py

```
#!/usr/bin/python

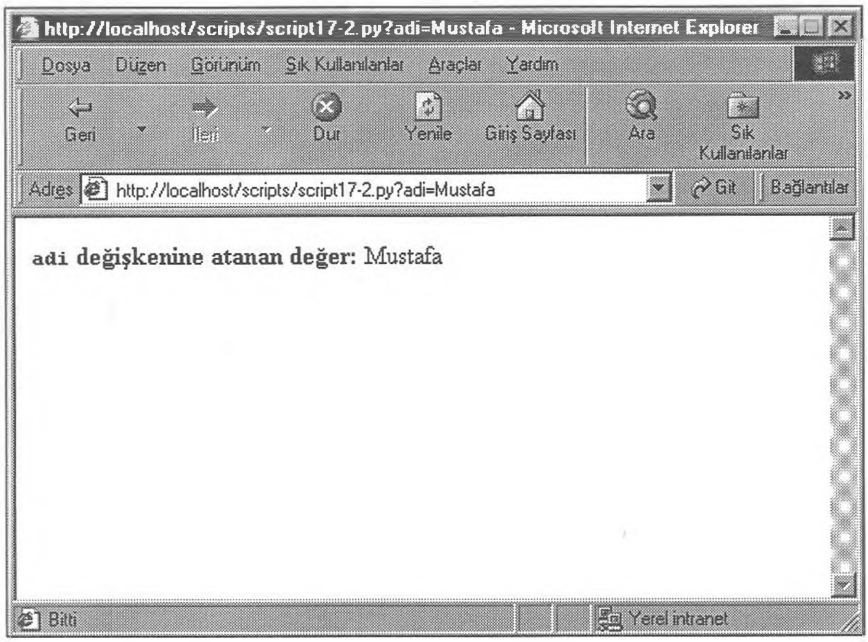
import cgi

form=cgi.FieldStorage()

print "Content-type: text/html"
print

print "<html>"
print "<body>"
print "<b><tt>adi</tt> değişkenine atanan değer:</b>"
print form["adi"].value
print "</body>"
print "</html>"
```

Bu scripti **http://localhost/scripts/script17-2.py?adi=Mustafa** ile çağırırsanız **Şekil-17.4**'deki gibi bir pencere ile karşılaşırsınız.



Şekil-17.4: Kullanıcı verilerinin alınması

Şimdi bir form oluşturalım ve bu formdan kullanıcı verilerini alalım. Bu amaçla aşağıdaki html kodunu yazıp web sunucunuzun ilgili yerine **html17-1.html** adıyla kaydediniz. (IIS için C:\inetpub\wwwroot).

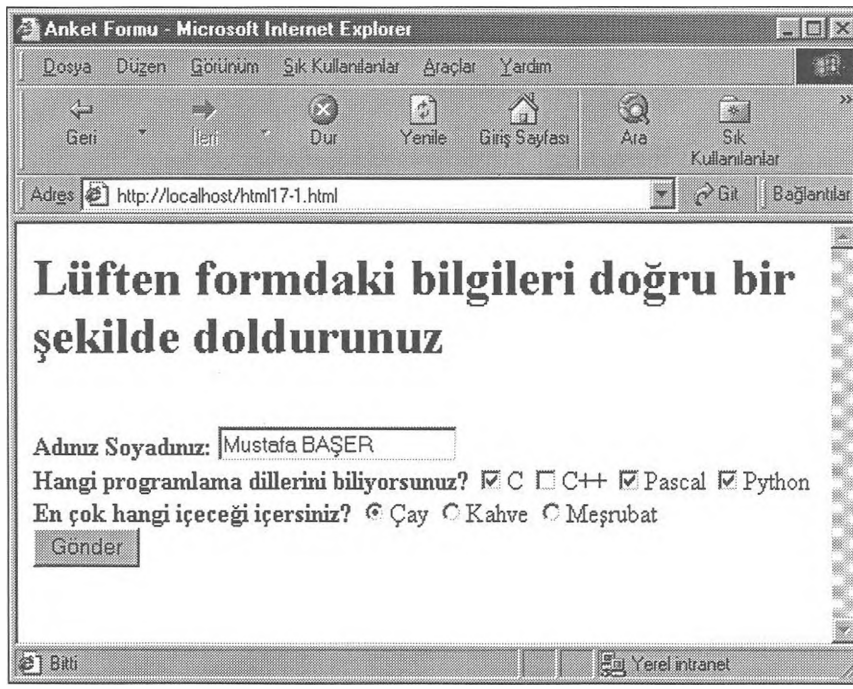
html17-1.html

```
<html>
<head><title>Anket Formu</title></head>
<body>

<form method="POST" action="/scripts/script17-3.py">
<h1>Lütfen formdaki bilgileri doğru bir şekilde doldurunuz</h1>
<br>
<b>Adınız Soyadınız:</b> <input type="text" name="adisoyadi">
<br>
<b>Hangi programlama dillerini biliyorsunuz?</b>
<input type="checkbox" name="dil" value="C">C
```

```
<input type="checkbox" name="dil" value="C++">C++
<input type="checkbox" name="dil" value="Pascal">Pascal
<input type="checkbox" name="dil" value="Python">Python
<br>
<b>En çok hangi içeceği içersiniz?</b>
<input type="radio" name="icecek" value="Çay">Çay
<input type="radio" name="icecek" value="Kahve">Kahve
<input type="radio" name="icecek" value="Meşrubat">Meşrubat <br>
<input type="submit" name="gonder" value="Gönder">
</body>
</html>
```

Bu html kodunun açıklamasını bildiğinizi varsayıyoruz. Bu sayfayı tarayıcımızdan isteyelim (**Şekil17-5**).



Şekil-17.5: Anket formu

Şimdi bu web sayfasındaki verilerin POST ile gönderileceği programı yazmaya çalışalım. Bu programda dikkat etmemiz gereken noktalar var:

- Kullanıcıdan gelen form alanının doldurulup doldurulmadığı (ya da işaretmenip işaretlenmediği) mutlaka kontrol edilmelidir. CGI modülü kullanıcıdan gelen verileri bir sözlükmüş gibi verdiğinden sözlükte olmayan anahtarı istersek hata oluştururuz. Html formalarında doldurulmamış ya da işaretlenmemiş alanlar programa iletilmezler.
- Bir alan ismi birden fazla değer taşıyabilir. Bunun için muhtemel çoklu değer alanlarını mutlaka kontrol edip, değerlerini ona göre almamız gerekiyor. Yukarıdaki "dil" alanı, eğer kullanıcı iki tane işaretlerse çoklu değer, bir tane işaretlerse tekli değerden oluşur. Bunların CGI modülünden alınması farklıdır.

Bu açıklamalar ışığında **script17-3.py**'yi şu şekilde yazmamız uygun olacaktır:

script17-3.py

```
#!/usr/bin/python
import cgi

form=cgi.FieldStorage()

print "Content-type: text/html"
print

print "<h1>Ankete katıldığınız için teşekkürler</h1>"
print "Ankete verdiğiniz cevaplar şöyle"

if form.has_key("adisoyadi"):
    print "<br><b>Adınız Soyadınız: </b>", form["adisoyadi"].value

if form.has_key("dil"):
    print "<br><b>Bildiğiniz Programlama dilleri: </b>"
    if type(form["dil"])==type([]):
```

```
        for e in form["dil"]:
            print e.value
    else:
        print form["dil"].value

if form.has_key("icecek"):
    print "<br><b>En çok sevdiğiniz içecek: </b>",
    form["icecek"].value
```

Bu programın çok karmaşık olduğunu düşünebilirsiniz.. Hatta Python ile CGI yazmanın çok zor olduğunu da düşünmeniz de normal. İlk intiba hemen hemen tüm insanlarda aynıdır (yazar da dahil). Ancak bütün bu işlemler maximum güvenlik için vardır. Bir değer gelmiş mi, gelmişse kaç değeri var gibi tüm denetimler yapılmalıdır. Güvenli bir site yapmak istiyorsanız şunu aklınızdan çıkartmayın: "Kullanıcıdan gelen bilgilere güvenmemeliyim". Aksi halde gözden kaçıracağınız ufak bir programlama yanlışı ile tüm sisteminizi çökerttirebilirsiniz.

Yukarıdaki betiği biraz tembel işine dönüştürelim. Bir fonksiyon yazarak gelen değerin var olup olmadığını, ne cinsten olduğunu belirleyen bir fonksiyon yazalım. Bunu bir modül haline getirip sonraki programlarımızın hepsinde içerelim:

mycgi.py

```
class CGI:
    def __init__(self, form):
        self.form=form

    def getVal(self,alan):

        if self.form.has_key(alan):
                if type(self.form[alan])==type([]):
                    donen=[]
                    for e in self.form[alan]:
                        donen.append(e.value)
```

```
                return donen
            else:
                return self.form[alan].value

        return ''
```

Şimdi **script17-3.py** betiğini şu şekilde yazabiliriz:

script17-4.py

```
#!/usr/bin/python

import cgi

form=cgi.FieldStorage()

from mycgi import CGI

alan=CGI(form)

print "Content-type: text/html"
print

print "<h1>Ankete katıldığınız için teşekkürler</h1>"
print "Ankete verdiğiniz cevaplar şöyle"
print "<br><b>Adınız Soyadınız: </b>", alan.getVal("adisoyadi")
print "<br><b>Bildiğiniz Programlama dilleri: </b>",
str(alan.getVal("dil"))
print "<br><b>En çok sevdiğiniz içecek: </b>", alan.getVal("icecek")
```

Saldırılara Dikkat!

CGI programlarına iki çeşit saldırı yapılabilir. Bunlardan birisi bir motor yazılarak size sürekli veri gönderilmesidir. Bu tür saldırılara **DOS attack** denmektedir. Bundan kurtuluş yolu yok gibidir. Her ne kadar web sunucuları aynı adresten gelen yoğun istekleri geri çevirebilmekte isede, ülkemizde bu-

lunan büyük proxy sunucularının varlığı nedeniyle bunu yapmak pek mantıklı gelmemektedir. Çünkü internete ulaşan kullanıcıların neredeyse yarıya yakını bu büyük proxy'lerden geçmekte ve sanki hep aynı adresten veri geliyor ya da istekte bulunuluyormuş gibi algılanmaktadır.

Bir diğer saldırı yöntemi ise web sunucunuza gerek ASP, gerek PHP ve gerekse CGI programları aracılığı ile yüksek miktarda veri gönderilmesidir. Bu veriler öylesine çok olmaktadır ki sunucunun hafızası ve diskini doldurmakta ve sunucu kilitlenmektedir. Bunu engellemek için mutlaka kullanıcıdan gelen verilerin boyutunu denetlemek gerekir. Siz eğe CGI modülünü çağırdıktan sonra gelmesini sitediğiniz en büyük veri miktarını belirtmeden **form=cgi.FieldStorage()** atamasını yaparsanız, kullanıcıdan gelen tüm veriler alınmaya çalışılır. Bunun için mutlaka maximum veri boyutunu belirtmelisiniz. Yukardaki scriptleri daha güvenli hale getirmek için CGI modülünü çağırdığınız satırın hemen altına

```
cgi.maxlen=100*1024
```

şeklinde bir satır eklemelisiniz. Bu satır sayesinde kulanıcıdan maksimum 100K'lık bilgi alınır. Bundan daha fazla bilgi göndermeye çalışırlarsa bir hata yükselerek program akışı durur.

Muhtemel diğer saldırı metodu da, scriptlerinizde bir hata yükseltmesi yaptırılarak sunucunun verdiği hata uyarılarının yakınlarında sistemle ilgili parolaları yakalamaktır. Bu yüzden parolaları kesinlikle (genellikle veritabanı) program içerisine yazmayınız. Web sunucusunun ulaşamayacağı bir yerde tutunuz ve program içerisinden okutunuz.

Çerezler (Cookie)

Kullanıcı size bağlantı yaptıktan sonra onu takip etmek, kimliğini denetlemek istiyorsanız bunu çerezler ile yapabilirsiniz. Bu amaçla her kullanıcıya has özel bir veri gönderirsiniz ve sonraki bağlantılarda bu veriyi alıp sizdeki ile karşılaştırırsınız. Eğer veri doğru ise kullanıcıya erişim için izin verirsiniz. Tüm bu özel veriyi iki yolla gönderip alabilirsiniz. Bunlardan ilki bu özel verinin hidden alanlar veya link eklemeleri ile sayfadan sayfaya dolaştırılması, diğeri ise çerez kullanmaktır. İkinci yol tembel web programcılarının tercihidir. Biz de bunu tercih edeceğiz.

Kullanıcıya özel bir veri üretmek iki şekilde yapılabilir:

- Uzunca bir rastgele sayı üretilir.
- O anki zaman verilir

Genelde ikincisi tercih edilir, çünkü oldukça kolay ve hemen hemen tekildir. Python saniyenin milyarda biri mertebesinde zamanı verebilmektedir. Bunun anlamı hiçbir zaman bir süreçte alınan zaman diğerindekine eşit olamaz:

```
>>> import time
>>> session="%f" % time.time()
>>> session
'1011825343.190000'
```

Çerezleri otomatik olarak kullanmak için Pyton'da **cookie** modülü vardır. Bunun kullanımı sanki bir sözlük kullanıyormuş gibidir. Şimdi kullanıcıya çerez atayan bir script yazalım:

script17-5.py

```
#!/usr/bin/python

import os, time, Cookie

eski=0
oturum=''

if os.environ.has_key("HTTP_COOKIE"):
    cookie=Cookie.SmartCookie()
    cookie.load(os.environ["HTTP_COOKIE"])

    if cookie.has_key("oturum"):
        oturum=cookie["oturum"].value
        eski=1
else:
```

```
    cookie=Cookie.SimpleCookie()
    oturum="%f" % time.time()
    cookie["oturum"]=oturum
    print cookie

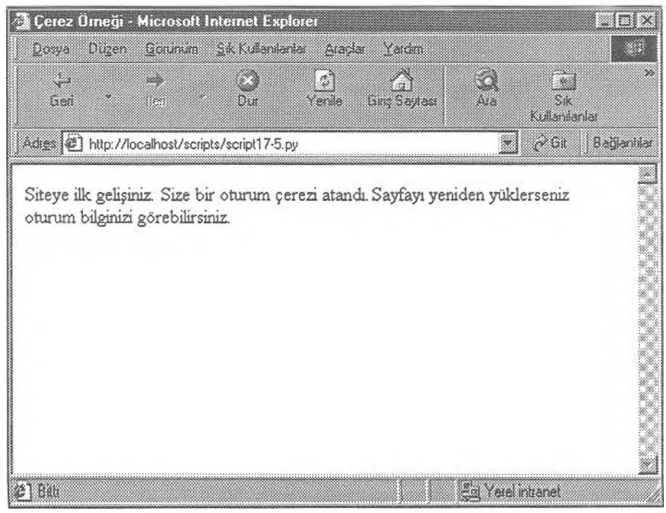
print "Content-type: text/html"
print

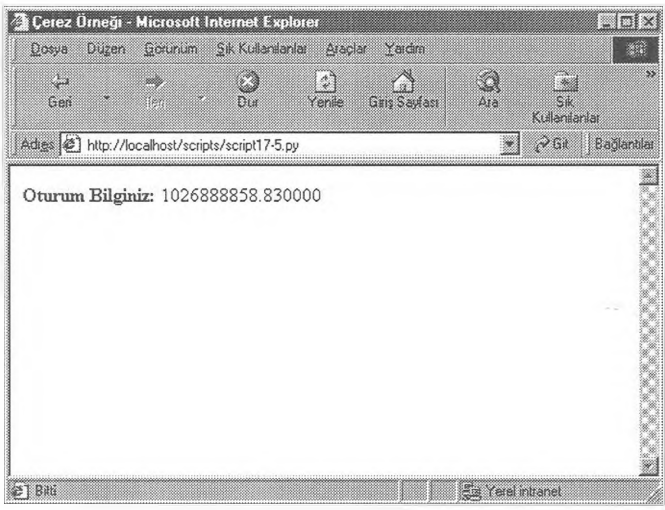
print "<html><head><title>Çerez Örneği</title></head><body>"

if eski:
    print "<b>Oturum Bilginiz: </b>", oturum
else:
    print """Siteye ilk gelişiniz. Size bir oturum çerezi atandı.
     Sayfayı yeniden yüklerseniz oturum bilginizi görebilirsiniz."""

print "</body></html>"
```

http://localhost/scripts/script17-5.py ile ilk erişimimizse çerezimiz olmadığı için bir açıklama metni gelecek. Eğer tarayıcımızın yeniden yükle düğmesine tıklarsak, ilk gelişimizde çerez atandığı için oturum bilgimiz yazılacaktır. **Şekil-17.6**'da bu çağrımları sıra ile görüyorsunuz.





Şekil-17.6: Çerezler Python ile oldukça kolay

Peki bu atadığımız çerezler ne işe yarar? Nasıl oluyor da biz kullanıcının eski kullanıcı olduğunu anlayabiliyoruz. Bunun için çerez değerleri mutlaka başka bir değerle eşlenerek bir veritabanına yazılır ve kullanıcıdan gelen değerlerle sizin veritabanınızda saklı bulunanlar karşılaştırılır. Diğer değer ya kullanıcı adıdır (bu da çereze atanır) ya da kullanıcının IP adresi olabilir. IP adresini çevre değişkeninden şu şekilde alabilirsiniz:

```
os.environ["REMOTE_ADDR"]
```

Dosya Yükleme

Python CGI modülünü kullanarak, sunucuya dosya yüklenmesini (file upload) sağlamak oldukça kolaydır. Önce kullanıcının dosyasını seçip yükleme yapabileceği basit bir html sayfası oluşturalım:

yukle.html

```
<html>
<head><title>Dosya Yükeleme</title></head>
<body>
<h1>Python Dosya Yükleme</h1>
<form method="POST" enctype="multipart/form-data"
action="/scripts/script17-6.py">
```

```
Sunucuya göndereceğiniz dosyayı seçiniz:
<input type="file" name="dosya">
<br>
<input type="submit" name="gonder" value="Yükle">
</body>
</html>
```

Bu sayfadan gelen verileri alacak olan Python scriptini yazalım:

script17-6.py

```
#!/usr/bin/python

import cgi, os

form = cgi.FieldStorage()

cgi.maxlen=500*1024

print "Content-type: text/html"
print

print "<html><head><title>Dosya Yükleme</title></head>"
print "<body>"

dosya = form["dosya"]

if dosya.filename:
    dosyaIsmi=os.path.basename(dosya.filename)

    print "<b><tt>",dosyaIsmi, "</tt></b> dosyasını yüklediniz."

    DOSYA=open("c:/tmp/"+dosyaIsmi,"w")
    DOSYA.write(dosya.file.read())
    DOSYA.close()
```

```
    print "<br>Bu dosya c:\tmp\%s olarak kaydedildi" % dosyaIsmi
else:
    print "Bir dosya seçtiğinizden emin misiniz?"

print "</body></html>"
```

Bu programda kullanıcının 500K'dan daha büyük dosyaları yükleyememesi için bir sınırlama koyduk (**cgi.maxlen**). Kullanıcının bir dosya gönderip göndermediğini, form elemanının **filename** özelliği ile öğreniyoruz. Eğer dosya gönderilmişse onu alıp diske kaydediyoruz, aksi halde kendisine dosya göndermediğini söylüyoruz. CGI programlarına dosya gönderilirken, kimi tarayıcılar sadece dosyanın ismini, kimileri ise dosyanın tam patikasını gönderir. Patikalar işletim sistemleri arasında farklılık gösterir. Ancak biz bunların hiçbirisi ile ilgilenmeden direk **os.path.basename(**<patika>**)** ile dosyanın ismini öğrenebiliriz. Dosya içeriğini almak için **dosya.file.read()** fonksiyonunu kullandık. Satır satır okumak için **dosya.file.readline()** kullanabilirsiniz. Dosya okunduktan sonra C:\tmp klasörüne, kullanıcının gönderdiği isimle kaydedildi. Web sunucusunun bu klasöre yazma yetkisi yoksa bir hata ile karşılaşırsınız.

Apache'de Kod Hataları

Yukarıda hep IIS kullanarak örnekleri verdik. Eğer sizde IIS kullanıyorsanız ve kod yazarken bir hata ile karşılaşmışsanız, bu hataların tarayıcıya gönderildiğini görmüşsünüzdür. Ancak Apache kullanıyorsanız sadece 404 mesajını görüp hatanın ne olduğunu göremezsiniz. Çünkü Apache standart hata çıktısını tarayıcıya göndermeyerek onu error.log dosyasına yazar. Eğer Apache'de de hataları tarayıcıya göndermek istiyorsanız kodda birkaç eklenti yaparak standart hatayı, standart çıktıya yönlendirmeniz gerekecektir. Scriptlerin genel yapısı şöyle olmalıdır:

#!/usr/bin/python

<modül çağrıları>

import traceback
sys.stderr = sys.stdout

```
<Diğer fonksiyonlar>

def main():
                    <script kodları>

try:
                    main()
except:
                    print "Content-type: text/html\n"
                    print "<pre>"
                    traceback.print_exc()
```

Bu sayede oluşabilecek tüm hatalar yakalanarak tarayıcıya gönderilmiş olur.

Linux'ta Diğer Web Programlama Metotları

Python kullanarak Linux üzerinde çeşitli şekillerde programlama yapabilirsiniz. Bunlardan ilki yukarıda anlattığımız CGI metodudur. Python'un mükemmelliği sayesinde birçok geliştirici onu web programcılığında çeşitli şekillerde kullanmıştır.

mod_snake ve PSP Python kodlarının html içerisine gömülmesi esasına dayanır. Kitabımızda mod_snake'den bahsetmezken PSP'yi ilk olarak anlatacağız.

Diğer bir metot ise **mod_python** kullanmaktır. Bununla aynen CGI programı yazarmış gibi dinamik sayfaları oluşturursunuz. Tek farkı son derece hızlı olmasıdır. Yazdığınız program sanki Apache'nin bir parçasıymış gibi davranır. Bu sayede performansta ulaşılabilecek en üst düzeye erişirsiniz. PHP'den sadece iki-üç kat bir hıza erişmeniz olası. **mod_python** ile web sayfaları geliştirmek biraz zahmetlidir ve kitabımızda bahsetmeyeceğiz.

Web geliştirme ortamının en üst düzeyini yakalamak istiyorsanız, Zope (Z Object Programming Environment) sizi bekliyor. Zope oldukça karmaşık, bir o kadar da mükemmeldir. İleri web programcılığı yapmak isteyenler *http://www.zope.org* adresini ziyaret ederek daha ayrıntılı bilgi alabilirler. Kendi sanal dosya sistemi ile, platformlar arası çalışabilen, güvenlik konu-

sunda herhangi bir taviz vermeyen Zope aynı zamanda Python'un geliştiricisi Guido van Rossom'un da çalıştığı şirketin bir ürünüdür. Hatta *"Artık Python sadece Zope'un ihtiyaçlarına yönelik geliştiriliyor."* eleştirisi bile almıştır kendisi. Kitabımızda Zope ile web uygulaması geliştirmeyi anlatmayacağız.

Linux'ta PSP

Linux'ta PSP kullanmak için yapmanız gereken bir dizi işlem var. Aşağıdaki anlatacaklarımız Red Hat 7.2 içindir. Diğer Linux dağıtımlarında da benzer işlemlerin geçerli olması gerekir. Eğer kullandığınız sistemin yöneticiliğini siz yapmıyorsanız, sistem yöneticinizden aşağıdaki işlemlerin yapılması rica etmeniz gerekiyor. Bu yöntemlerin herhangi bir UN*X'te de çalışması beklenir.

Öncelikle Apache'nin PSP'leri yorumlayabilmesi için gömülü bir istemciye ve bu istemciye cevap verecek bir sunucuya ihtiyaç vardır. Şunu hemen belirtelim ki PSP'yi Linux'ta kullanmak perfomansı oldukça düşürür (Bununla birlikte CGI'nın 2-3 katı kadar bir hız sahibi olursunuz). Eğer Linux platformunda Python ile Web programları yazacaksanız en iysi **mod_python** kullanmak olacaktır. **mod_python**'un en hızlı web yazılım dili olarak bilinen PHP'den birçok işlemlerde en az birkaç kat daha hızlı olduğunu belirtelim. Aynı PSP Windows/Apache üzerine kurularak çalıştırılabilir. Ancak Windows kullanıyorsanız IIS/ASP/Python'u öneririz (İlerki bölümlerde anlatacağız).

PSP istemci ve sunucusunu şu adresten indirebilirsiniz:

```
http://webware.sourceforge.net/
```

Kitabımızın bu bölümünün yazıldığı sırada son kararlı sürümü şu adresten indirdik:

```
http://prdownloads.sourceforge.net/webware/Webware-0.7.tar.gz
```

Bu dosyayı ev dizinimize açalım:

```
# tar -zxf Webware-0.7.tar.gz
```

İlk olarak Apache modülünü derlememiz gerekiyor. Bunun için:

```
# cd ~/Webware/WebKit/Native/mod_webkit
# make
```

Bu modülü Apache modüllerinin bulunduğu klasöre kopyalayalım:

```
# cp mod_webkit.so /etc/httpd/modules/
```

Webkit'i kurmanız gerekiyor:

```
~/WebWare/install.py
```

Şimdi de **/etc/httpd/conf/httpd.conf** dosyasını herhangi bir metin düzenleyici ile açıp (vi, pico, nedit, kwrite gibi...), dosyanın en sonuna aşağıdaki satırları yazınız:

```
LoadModule webkit_module modules/mod_webkit.so
AddModule mod_webkit.c

AddType text/psp .psp
AddHandler psp-handler .psp

<Location /WKMod>
WKServer localhost 8086
SetHandler webkit-handler
</Location>
```

Apache'yi yeniden başlatmadan önce PSP ile sonlanan dosyaları sistemde mime tipi olarak belirtmek gerekir. Bunun için **/etc/mime.types** dosyasını açıp en sonuna şu satırı ekleyin

```
text/psp                        psp
```

Şimdi web sunucumuzu başlatabiliriz.

```
# /etc/init.d/httpd restart
```

Son olarak PSP sunucusunu çalıştırmak gerekiyor. Bu amaçla şu komutu işletin:

```
# ~/Webware/WebKit/AppServer
```

Eğer her şey yolunda ise sunucu başlatıldığında son satırlar şöyle olmalıdır:

```
...
...
...
Loading plug-in: TaskKit at /root/Webware/TaskKit
Loading plug-in: COMKit at /root/Webware/COMKit
    Plug-in /root/Webware/COMKit cannot be loaded because:
    Required op sys is ['nt'], but actual op sys is posix.

Listening on ('127.0.0.1', 8086)
Creating 10 threads..........
Ready
```

Hemen bir bir PSP sayfası oluşturalım. Bu amaçla çok kullandığınız bir metin düzenleyiciyi açın ve aşağıdaki satırları yazın:

```
<HTML>
<BODY bgcolor='white'>

<h1 align=center>Bu bir PSP Sayfasıdır</h1>

Şimdi <b><tt>/root/Webware</tt></b> Klasöründeki
dosyaları bir tablo içinde listeleyelim:
<p>
<table border=1>
<tr><td><b>Dosya   Adı</b></td><td><b>Son   değiştirme   Tari-
hi</b></td></tr>

<%
import os, time
```

Python

```
klasor="/root/Webware/"

for d in os.listdir(klasor):
                    res.write("<tr><td>%s</td><td>%s</td></tr>"   %
(d, time.ctime(os.stat(klasor+d)[8])))

%>

</table>

</BODY>
</HTML>
```

Bu dosyayı **ornek.psp** ismi ile **/var/www/html** klasörüne kaydedin.

Burada gördüğünüz üzere normal bir web sayfası hazırlıyormuş gibi başladık. Python kodlarını ise **<% %>** tagları içerisine yazdık. Tek farkı **print** fonksiyonu yerine PSP handler olan **res.write**'ı kullanmamızdır.

Şimdi web tarayıcınızı açıp adres satırına **http://localhost/ornek.psp** yazın. Karşılaşacağınız ekran **Şekil-17.7**'deki gibi olacaktır:



Şekil-17.7: Linux'ta ilk PSP'nin çalıştırılması.

PSP sunucusunu çalıştırdığınız ekranda olup bitenleri takip edebilirsiniz. Örneğin yukarıdaki süreçler için şöyle bir çıktı görürsünüz:

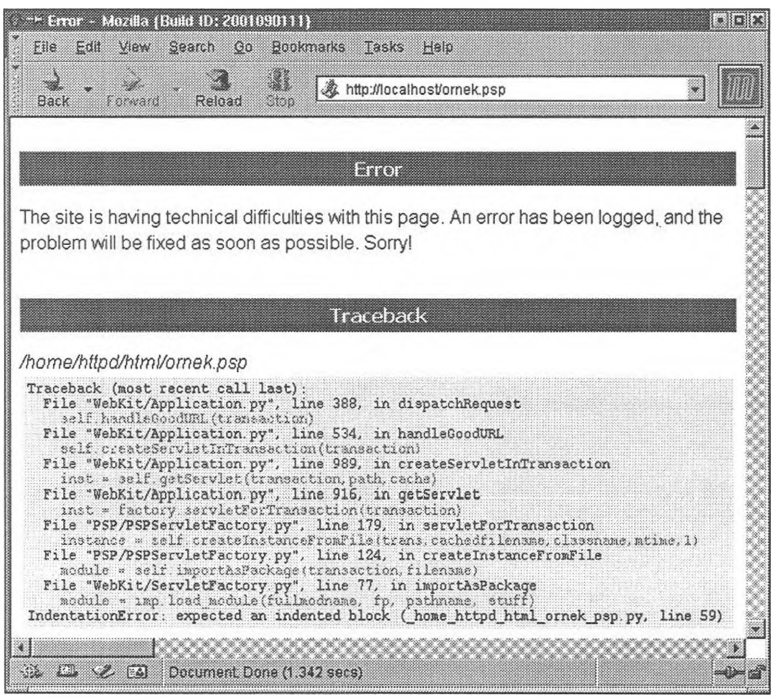
```
29  2002-03-04 17:03:08  /ornek.psp
29  0.01 secs            /ornek.psp
```

Eğer PSP sunucsunun sistemin her açılışında başlamasını istiyorsanız bunu **/etc/rc.d/rc.local** dosyasına şu şekilde belirtebilirsiniz:

```
/root/Webware/WebKit/AppServer > /var/log/pspServer.log &
```

Bu yazılımda, ekrana yazılan süreçler de, /var/log/pspServer.log dosyasına yazılmış olacaktır. Burada çok dikkatli olmalısınız, PSP sunucu root hakları ile çalışıyor ki bu son derece tehlikeli. En iyisi başka bir kullanıcı adı ile başlatmak olacaktır. Konumuz UN*X olmadığından bunun nasıl yapılacağınız sistem yöneticilerine bırakıyoruz.

Olur ya kod yazılımında bir hata yaparsanız, PSP server size **Şekil-17.8**'deki gibi son derece anlamlı bir pencere gösterecektir.



Şekil-17.8: PSP sunucusu hatalarınızı son derece açık ve seçik size bildirir

PSP'de form bilgilerini almak için **req._fields** kullanılır. Aşağıdaki programda adi değişkenine verilen bilginin nasıl alındığını görüyorsunuz.

ornek2.psp

```
<HTML>
<BODY bgcolor='white'>

<h1 align=center>Bu bir PSP sayfasıdır</h1>

<b><tt>adi<tt> değişkenine atanan değer: </b>

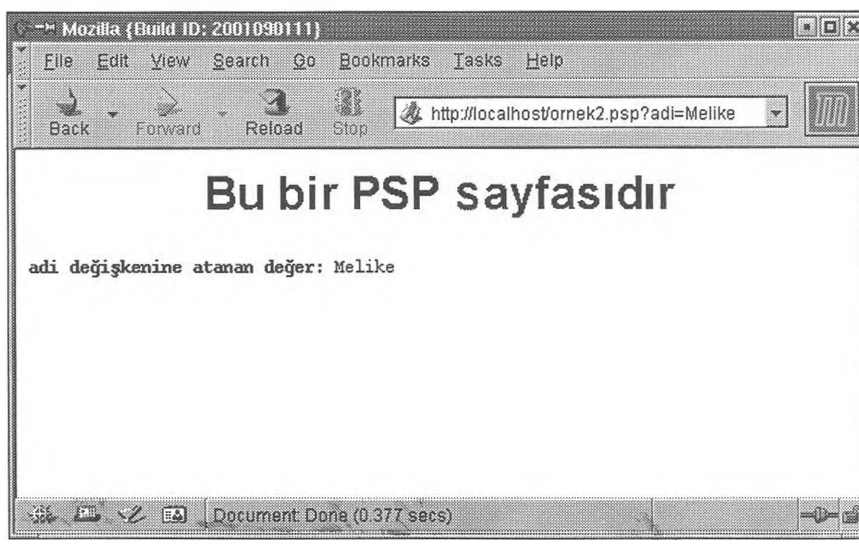
<%

res.write(req._fields["adi"])

%>

</BODY>
</HTML>
```

Bu sayfayı çağırdığımızda **Şekil-17.9**'daki gibi bir pencere elde edersiniz.



Şekil-17.9: PSP'de kullanıcıdan gelen verilerin alınması

PSP ile anlatacaklarımız bu kadar. Elbette PSP'nin mahareti bizim anlattıklarımızla kalmıyor. Cookie'lerin otomatik yönetimi, session takibinin tamamen otomatikleşmesi, kullanıcı onaylaması, kalıcı SQL bağlantıları gibi birçok özelliği vardır. Ayrıntılı bilgi için indirdiğiniz kaynak kodu içerisindeki dökümanlara bakmalısınız.

IIS'de Python ile ASP

ASP'nin bir dil değil teknoloji olduğunu söylemiştik. Eğer Windows platformunu kullanıyorsanız, IIS'de çalışırken büyük bir ihtimalle birkaç ASP kodunu incelemiş olmalısınız. ASP VBScript kullanmaktadır. VBScript Visual Basic nesnelerinin sadeleştirilmiş halidir. Ne kadar basit olduğu söylensede Python'u bilenler ASP/VBScript ile web sayfaları geliştirmenin hiç de öyle lanse edildiği kadar kolay olmadığını açıkça görecektir. Bu bölümde ASP'de VBScript yerine nasıl Python kullanacağınızı, dinamik web sayfalarının ne kadarda kolay ve kısacık kod yazılımı ile geliştirebileceğinizi göreceksiniz.

ASP/Python'un performansının ASP/VBScript'ye göre 3-4 kat olması sizi şaşırtmamalıdır.

Python'u ASP'de kullanabilmek için öncelikle onu kaydetmelisiniz. Registery ayarlarında oldukça karmaşık işler yapılmalıdır. Ancak Python geliştiricileri bunu otomatiğe bağlamışlardır. Kayıt işleminin yapılabilmesi için, şu Python porgramını çalıştırın:

```
C:\Python22\win32comext\axscript\client\pyscript.py
```

Eğer herşey yolunda gitmiş ise, program ekranınıza

```
Registered: Python
Registration of Scripting Engine complete.
```

çıktısını yazmalıdır.

İlk ASP programımızı yazalım:

python1.asp

```
<%@ LANGUAGE = Python%>

<html>
<head><title>ASP/Python</title></head>
<body>
<h1>Python Server Pages</h1>

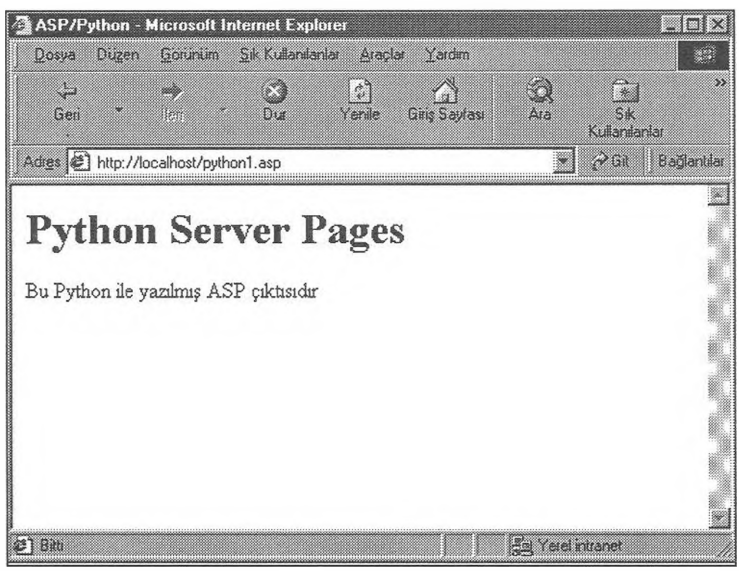
<%

Response.Write("Bu Python ile yazılmış ASP çıktısıdır".)

%>

</body>
</html>
```

Bu kodu yazdıktan sonra web sunucunuzun ana klasörüne **python1.asp** olarak kaydedin (IIS'nin ana klasörü genellikle C:\inetpub\wwwroot'dur). Şimdi tarayıcınızın adres satırına *http://localhost/python1.asp* yazınız. Açılacak olan sayfanın benzeri **Şekil-17.10**'dadır.



Şekil-17.10: İlk ASP/Python

Bu çalıştırıldığında birçok kişiyi hayretler içerisinde bırakmaktadır. Python ile program yazmanın ne kadar kolay olduğunu biliyorsunuz, bu şimdi ASP ile birleşince web sayfası geliştirme işi çocuk oyuncağına dönüyor.

Python ile ASP sayfaları geliştiriken ilk satırın mutlaka

```
<%@ LANGUAGE = Python%>
```

olması gerekiyor. Bu satır, bu sayfanın yorumlayıcısının Python olduğunu web sunucusuna söyleyerek, sayfanın Python'a gönderilmesini sağlıyor. Python kodları <% %> tagları arasına yazılmalıdır. İstediğiniz kadar Python fonksiyonu, sınıfı (kısacası nesneleri) tanımlayabilirsiniz veya import ile çağırabilirsiniz. Taglar dışında html kodları yazıp, daha önceki taglarda tanımladığınız fonksiyon ya da çağırdığınız modülleri kullanabilirsiniz.

Şimdi yukarıda, Linux/PSP'de yaptığımız örneği ASP/Python ile tekrarlayalım:

python2.asp

```
<%@ LANGUAGE = Python%>
<HTML>
<BODY bgcolor='white'>

<h1 align=center>Bu bir ASP/Python Sayfasıdır</h1>

Şimdi <b><tt>c:\Python22</tt></b> Klasöründeki
dosyaları bir tablo içinde listeleyelim:
<p>
<table border=1>
<tr><td><b>Dosya Adı</b></td><td><b>Sondeğiştirme
Tarihi</b></td></tr>

<%

import os, time
```

Python

```
klasor="c:/python22/"

for d in os.listdir(klasor):
      Response.write("<tr><td>%s</td><td>%s</td></tr>" % (d,
time.ctime(os.stat(klasor+d)[8])))

%>

</table>

</BODY>
</HTML>
```

Bu ASP sayfasını çağırdığımızda, **Şekil-17.11**'deki pencere ile karşılaşıyoruz.



Şekil-17.11: ASP/Python ile c:\Python22 klasörünün web sayfası halinde sunulması

Gördüğünüz gibi ASP/Python ile bildiğiniz şeyler artık daha basit.

Kullanıcıdan gelen verileri almak da oldukça kolay. GET metodu ile gelen verileri almak için **Request.QueryString(<alan>)**, POST ile gelenleri almak için ise, **Request.Form(<alan>)** kullanabilirsiniz. Yine yukarıda verdiğimiz örneği tekrarlayalım ve adi değişkeninin değerini alalım:

python3.asp

```
<%@ LANGUAGE = Python%>

<HTML>
<BODY bgcolor='white'>

<h1 align=center>Bu bir ASP/Python Sayfasıdır</h1>

<b><tt>adi<tt> değişkenine atanan değer: </b>

<%

Response.Write(Request.QueryString("adi"))

%>

</BODY>
</HTML>
```

Program çağırıldığında **Şekil-17.12**'deki gibi olmalıdır.



Şekil-17.12: ASP/Python ile kullanıcı verilerinin alınması

ASP/Python'da daha başka nesneler de vardır. Bunlar Request, Response, Session, Application, and ObjectContext'dir. Bunların hepsi ile ilgili örnek veremeyeceğiz.

HTMLGen

Pythonun web sayfalarını oluşturmada ne kadar maharetli olduğunu gördünüz. Her ağıza göre bal vardır. Eğer html kodlarını üretmek size zor geliyorsa, nesne modelli html üretmeye ne dersiniz? Bu amaçla geliştirilmiş olan HTMLGen'i Python ana sayfasından bulabilirsiniz (*http://www.python.org*). İlk başlarda öğrenmesi biraz sıkıcı gelse de, sonraları sizin üretkenliğinizi artımakta, birçok gereksiz kod yazılımından kurtarmaktadır. Yukarıda verdiğimiz C:\Python22 kalsöründeki dosyaları tablo ile verme örneğini HTMLGen ile yazalım:

python4.asp

```
<%@ LANGUAGE = Python%>

<HTML>
<BODY bgcolor='white'>

<h1 align=center>Bu bir ASP/Python Sayfasıdır</h1>

Aşağıdaki tablo HTMLGen ile üretilmiştir

<%

import os, time, HTMLgen

klasor="c:/python22/"

Table=HTMLgen.Table('C:\Python22 klasöründeki dosyalar')
Table.heading=('Dosya Adı','Son değiştirme Tarihi')
for d in os.listdir(klasor):
        Table.body.append([d,time.ctime(os.stat(klasor+d)[8])])

Response.Write(str(Table))

%>
```

HTMLgen'i diğer tüm web programlama yöntemlerinde kullanabilirsiniz. Bu programın tarayıcıdaki görüntüsü Şekil-17-13'de görünüyor.



Şekil-17.13: HTMLGen ile web sayfaları daha kolay oluşturulabilir.

İNTERNET TEKNOLOJİLERİ

ZAFER DEMİRKOL / 272 SAYFA



İnternet dediğimiz dev sistem her saniye milyarlarca işlem yaptığı halde nasıl kusursuz çalışıyor? Sistemi neler oluşturuyor? İnternette hangi teknolojiler ve cihazlar hangi amaçlarla kullanılıyor? Milyonlarca bilgisayar birbirleriyle nasıl iletişim kurabiliyor? Hemen her gün karşımıza çıkan bir konu olmasına rağmen internetin nasıl çalıştığını ve hangi temellere oturduğunu hiç merak ettiniz mi?

"Evet" diyorsanız, bu kitap tam size göre.

XML

432 SAYFA / 2. BASKI

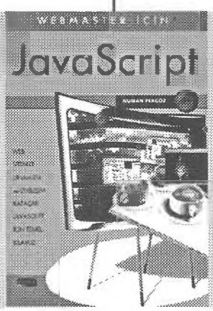


Daha önce Pusula Yayıncılıkdan çıkan ve satış listelerinde *best-seller* edebiyat kitaplarını bile geride bırakan *ASP ile Web Programcılığı ve Elektronik Ticaret* ve *İnternet Teknolojileri* adlı kitapların yazarı Zafer Demirkol, XML'i en başından itibaren anlatıyor. Demirkol'un kitabı sadece XML değil, onunla birlikte gelen veya kullanılan DTD, Schema, CSS, XSL, XSLT gibi teknolojileri de detaylı olarak kapsıyor.

Zafer Demirkol XML'i adım adım uygulamalarla, 200'ü aşkın örnekle yalın ve hedefe yönelik bir dille anlatıyor. Bir kullanıcının XML'i ve birlikte gelen teknolojileri öğrenmesi için basit bir HTML bilgisi olması, Demirkol'un anlatımını izlemesi ve örnekleri uygulaması yeterli.

JavaScript

NUMAN PEKGÖZ / 264 SAYFA / 2. BASKI



Sayısız web sitesinin tasarım ve programlamasında imzası bulunan, yurtiçinde ve yurtdışında çeşitli kurumlarda çalışarak web mühendisliği alanında uzmanlaşan Numan Pekgöz'ün kaleme aldığı *JavaScript*, bu programlama diliyle daha önce hiç tanışmamış ve ya tanışıp da yeterli ve düzeyli bir kaynak bulamamış okurları hedef alıyor, web sayfalarına, ihtiyacı olan dinamizm ve etkileşimi katmak isteyen amatör ve profesonel tüm web tasarımcı ve web-geliştiricilerini belirli bir düzeye getirmeye çalışıyor. Konusunda belli bir çizgiye ulaşmış olan programcılar da, Numan Pekgöz'ün kaleminden pek çok örneği inceleyip özgün uygulamalar bulabilecekleri pratik bir başvuru kaynağına kavuşacaklar.

DHTML ve CSS

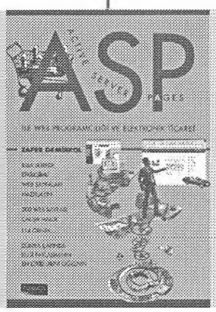
NUMAN PEKGÖZ / 504 SAYFA



İnternet artık hem iş hayatımızın hem de özel hayatımızın kalıcı bir parçası haline geldi. Ama günlük hayata girip belirleyici olan her teknoloji gibi o da sürekli değişiyor ve gelişiyor. DHTML bugün webmaster'ların en çok tercih ettiği İnternet teknolojilerinden biri. Numan Pekgöz'ün kaleme aldığı *DHTML ve CSS* işte bu teknolojinin bütün inceliklerini anlatıyor. Web tasarımı konusunudaki deneyimlerini pek çok site yapımında ortaya koyan Pekgöz, *DHTML ve CSS*'de en karmaşık konuları bile sade bir dille okuyucusuna aktarıyor.

ASP ile Web Programcılığı

ZAFER DEMİRKOL / 424 SAYFA / 6. BASKI



Web programcılığı ve elektronik ticaret teknolojisinin vazgeçilmez dili ASP konusunda kapsamlı bir kaynak kitap... ASP teknolojisi, günümüzün profesyonel web tasarımcılarına çok kısa sürede, düşük maliyetle, güçlü ve etkileşimli web sayfaları oluşturma imkânı veriyor.

ASP ile Web Programcılığı ve Elektronik Ticaret az laf, bolca örnekten oluşuyor. Kitap yaklaşık 200 web sayfası ve çalışan 114 örnek içeriyor. ASP öğrencileri her örnekteki kodları yazıp test edebilecekler, kendi yazdıklarını www.pusula.com adresindeki çalışan örneklerle karşılaştırabilecekler.

FRONTPAGE 2000

CENK TARHAN / 272 SAYFA / 2. BASKI



Kendi web sayfalarınızı hazırlarken HTML kodları ile uğraşmak istemiyorsanız bir HTML editörü elinizin altında olmalı. HTML editörleri arasında kullanımı en kolay programlardan biri olan FrontPage 2000 bu açıdan acemi web master'ların en büyük yardımcısı. Bu kitapta FrontPage 2000 programının kurulumundan kullanımına kadar pek çok detayı ve püf noktasını bir arada bulabileceksiniz.

WINDOWS XP

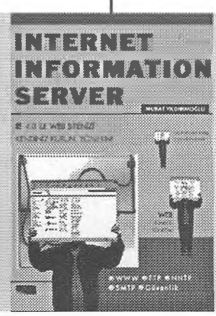
CENK TARHAN / 248 SAYFA / 2. BASKI



Windows XP artık bir standart haline gelen Windows işletim sisteminin en son sürümü. Windows XP eski Windows ailesinden gelen Windows 98 ve ME gibi işletim sistemlerinin kullanım kolaylığı, donanım desteği ve pratikliği ile ofisler ve kritik işlemlere yönelik NT işletim sisteminin güvenilirliğini, performansını sağlam bir sistem mimarisi altında birleştirmeyi başardı. Bu yönüyle Windows XP, önümüzdeki yıllarda hemen hemen tüm Windows kullanıcılarının sıçramak isteyeceği başarılı bir işletim sistemi olacak. Kitap, yeni işletim sistemiyle gelen bütün özelliklerin üzerinde tek tek duruyor ve kullanıcılara, yeni işletim sisteminin imkânlarından en iyi şekilde yararlanabilmeleri için gerekli püf noktalarını açıklıyor.

INTERNET INFORMATION SERVER

MURAT YILDIRIMOĞLU / 144 SAYFA

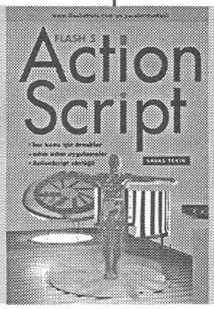


Internet Information Server (IIS) son zamanların en popüler web sunucusu. Murat Yıldırımoğlu'nun kitabı, kolay erişilebilir ve güvenli web sayfaları hazırlamanıza imkân veren bu programı anlatıyor. IIS nasıl kurulur, nasıl yapılandırılır, sorunları nasıl çözülür gibi temel soruların yanında, bu konunun kavramsal temellerini oluşturan diğer açıklamalar...

Daha önce Pusula Yayıncılıktan altı kitabı çıkan Yıldırımoğlu en çok *TCP/IP* kitabıyla tanınıyor. Bilgisayar alanındaki tecrübelerini başkalarına etkin bir dille aktarabilme tecrübesiyle de geliştiren yazar, *Internet Information Server* kitabında da bu çağdaş web teknolojisini, konuya yeni adım atanların sorabileceği soruları göz önüne alarak yazdı.

ActionScript

SAVAŞ TEKİN / 744 SAYFA



Web tasarımı denince akla ilk gelen uygulamalardan biri Flash'tir. Flash ilk çıktığı günden bugüne popülerliğini artırdı ve hemen hemen her sitenin Flash versiyonu yapıldı.

Özellikle Flash'ın 5 sürümüyle birlikte artık bir script dili haline gelen ActionScript, sadece web sayfası tasarımında değil, bilişim sektörünün diğer birçok kolunda da büyük bir çığır açmış ve profesyonel uygulamalar için vazgeçilmez olmuştur.

Perl ve MySQL ile CGI

CAN UĞUR AYFER / 360 SAYFA



İnternet'e, kullanıcıların seçimlerine ve gereksinimlerine göre dinamik olarak düzenlenen web sayfaları hâkim olmaya başladı. Bu da doğal olarak, bu işi yapabilmek için arka planda çalışacak birtakım yazılımların ve bunlarla ilgili standartların geliştirilmesi gereksinimini ortaya çıkardı.

Bu standartlardan en önemlilerinden birisi CGI (Common Gateway Interface) ile programlamadır. CGI programlamanın iki incisi var: Perl ve MySQL. Her ikisi de "Açık Kaynak" yazılımı; yani bedel ödemeden rahatça kullanılabilen, kusursuz ve mükemmel performansla çalışan yazılımlar.

PERL CGI

ÖZCAN ACAR / 239 SAYFA

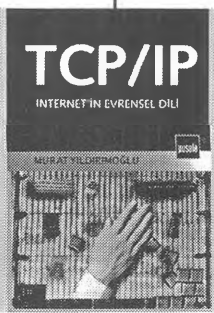


Web sitelerinde çok değişik amaçlarla kullanılan bir programlama yöntemi hakkında temel bir kaynak. CGI (Common Gateway Interface), HTML sayfalarını dinamikleştirmek için kullanılan bir programlama standardıdır. CGI'ın en çekici özelliği, web sitesi ziyaretçisiyle diyaloğa girmesidir. CGI ile kullanıcının istekleri doğrultusunda dinamik HTML sayfaları oluşturulur, işlemler yapılır, e-posta gönderilir. Bu yöntemin en çok kullanılan programlama dili ise Perl'dür.

Perl, CGI Pusula Yayıncılıktan çıkan bütün kitaplar gibi, doğrudan doğruya Türkçe okurlar için yazılmış, özgün örnekler içeren vazgeçilmez bir kaynak.

TCP/IP

MURAT YILDIRIMOĞLU / 148 SAYFA / 3. BASKI



İnternetin temelinde yatan protokol olan TCP/IP, internet ile birlikte giderek büyük bir popülerlik kazandı. İnterneti sadece gezgin olarak değil, katılımcı olarak kullanmayı düşünüyorsanız bu protokolün nasıl çalıştığını ve prensiplerini de öğrenmeniz gerekiyor.

Türkçede kaynak sıkıntısı çekilen bir konuda, deneyimli bir yazar tarafından kaleme alınan bu kitap önemli bir eksikliği gösteriyor. TCP/IP'nin genel yapısının yanı sıra IP Adresleme, DNS, WINS isim sistemleri, SNMP ve DHCP protokollerinin kullanımı gibi konularda da bilgi veren kitap, kullanım örnekleriyle de destekleniyor. Sadece internet değil, TCP/IP protokolünün kullanıldığı tüm ağlar üzerinde çalışanlar için vazgeçilmez bir başvuru kaynağı..

Python



- Python çok amaçlı, nesneye yönelik modüler bir programlama dilidir.
- Python'da sistem programcılığı için gerekli her türlü araç mevcuttur.
- Python Windows uygulamalarıyla tam uyumludur.
- Linux/UNIX için birçok özel kütüphaneye sahiptir.
- GUI programcılığı için idealdir. Birçok GUI toolkit'i barındırır.
- Veritabanı programcıları için her türlü veri sistemine rahatça erişebilecekleri kütüphaneleri içerir.
- Web programcıları Python ile daha hızlı, daha kolay ve daha etkili siteler oluşturabilirler.
- Birçok portal, hizmetlerinde Python programları kullanmaktadır.
- Python, .net için kullanılmaya hazır dillerin öncüsüdür.

MUSTAFA BAŞER

1970 yılında Yozgat'ta doğdu. ODTÜ Fizik Öğretmenliği bölümünü bitirdi. Yüksek lisans çalışmalarını yine aynı üniversitenin Fen Bilimleri Eğitimi bölümünde yaptı. Fen ve matematik eğitimi dalında doktora çalışmalarına devam etmektedir. Bilişim sektöründe ve çeşitli eğitim kurumlarında çalışmakta, CHIP, PC Magazin, PCNET ve PC World gibi popüler bilgisayar dergilerine Python ve Linux üzerine yazılar yazmaktadır. Ayrıca Linux, Windows NT, Python, Samba gibi konularda ders vermektedir. Linux ve Python ile uğraşmadığı zamanlarda mangal yakmak, kırlarda gezintiye çıkmak, kitap okumak gibi hobiler geliştirmiştir. Eşi Dilek'ten Fatih ve Melike isimli iki çocuğu vardır.

ISBN 975-7092-99-1

9 789757 092995

20.00₺

PUSULA