Python Programlama Dili Eğitimi

Ege GNU/Linux ve Özgür Yazılım Çalışma Grubu
www.ozgurpenguen.org

CC BY SA

Enes Ateş
enes@enesates.com

İyte - Sem, İzmir, 2013

python™

İçerik

- “Merhaba Python!”
- Kontrol Yapıları
- Veri Yapıları
- Modüller
- Dosya İşleme
- Hata Yakalama
- Sınıflar
- Pokemon Oyunu

Konu içeriklerine erişmek için konu başlıklarına tıklayabilirsiniz.

python™

“Merhaba Python!”

Python Nedir?
Nerelerde Bulunur?
Nasıl Kullanabilirim?
>>>
Değişkenler
Kullanıcıyla İletişim
.py
Vim

python™

Python Nedir?

- Özgür yazılım
- Monty Python's Flying Circus - Guido van Rossum
- Basit ve şık söz dizilimli:
  - Girintileri sever, süslü parantezlerden kaçınır.
  - Okunabilirlik önemlidir.
- Yüksek seviyeli
- Yorumlanan (interpreted) programlama dili
- Fonksiyonel, prosedürel ve nesneye dayalı programlama paradigmaları kapsamında

python™

Python Nedir?

- Dinamik tip tanımlı
- Güçlü veri yapısı desteği
- Birimsel (modüler)
- Piller dahil (fazla sayıda modül içeren standart kütüphane)
- Betiklerden (script) büyük projelere
  - Sistem, kullanıcı arabirimi, ağ, veritabanı programlama vb.
- Çoklu platform desteği (GNU/Linux, Windows, Mac OS vb.)

python™

Nerelerde Bulunur?

- Blender, GIMP, Inkscape
- Ubuntu, Fedora, Gentoo, Pisi GNU/Linux
- Django Web Framework
- Apache,
- BitTorrent,
- Google, Yahoo, Facebook
- CERN, NASA
- Çoğu GNU/Linux dağıtımında öntanımlı bileşen

python™

Nasıl Kullanabilirim?

- GNU/Linux dağıtımlarında:
  - Uçbirim (terminal) üzerinden
    - ***python*** komutuyla
- Windows ortamında:
  - *www.python.org/download* uygun sürüm indirilir.
  - Python YOL'a (PATH) kurulu olduğu dizin değeriyle eklenir. tinyurl.com/pythonyol
  - Komut satırından ***python*** komutuyla kullanılabilir.
- *Eğitim içeriğinde Python 2.x sürümü referans alınmıştır (2.7.3)*

python™

>>>

```
>>> 2 + 2
4
>>> (50 – 5 * 6) / 4  # Bu bir yorum satırıdır.
5
>>> 7 / 3    # Tam sayıların bölümünün sonucu aşağıya (floor) yuvarlanır.
2
>>> 7.0 / 2
3.5
>>> 5 ** 3   # 5'in 3. kuvvetini alır.
125
```

python™

>>>

```
>>> x = y = z = 0
>>> x
0
>>> y
0
>>> z
0
>>> a   # tanımlanmamış değişkene erişmeye çalışmak
NameError: name 'a' is not defined
```

python™

>>>

```
>>> genislik = 22.4  ;  yukseklik = 5.3 * 9.7   #  ;  karakteri satırları tek satırda birleştirir
>>> genislik * yukseklik
1151.5839999999998
>>> alan = _    # en son üretilen değeri alır.
>>> alan
1151.5839999999998


>>> round(alan, 2)   # hassasiyeti ayarlar.
1151.58
```

python™

>>>

```
>>> 'ozgur yazilim'   # karakter dizileri için tek ya da çift tırnak kullanılabilir.
'ozgur yazilim'
>>> “ozgur yazilim”
'ozgur yazilim'


>>> 'Asimov'un Vakıf dizisi'   # karakter dizisinde tırnak işareti kullanmak
SyntaxError: invalid syntax    # sözdizim hatası
>>> “Asimov'un Vakıf dizisi”   # farklı tipte tırnak işareti kullanmak gerekir
>>> 'Asimov\'un Vakıf dizisi'   # kaçış karakterlerinden ters bölü de kullanılabilir.
```

python™

>>>

```
>>> il = 'izmir'
>>> ilce = 'bornova'
>>> adres = il + '/' + ilce  # karakter dizileri + operatörü ile birleştirilebilir (concatenation).
>>> adres
'izmir/bornova'

>>> print  adres  # karakter dizisi yazdırmak için print fonksiyonu kullanılabilir.
izmir/bornova
>>> print  'Yasadigi il = %s \nYasadigi ilce = %s'  %(il, ilce)   # formatlı çıktı
```

python™

>>>

```
>>> en_uzun_kelime = \        # \ karakteri uçbirimde çalışırken alt satıra geçmeyi sağlar.
. . .  'muvaffakiyetsizlestiricilestiriveremeyebileceklerimizdenmissinizcesine'
>>> uzunluk = len(en_uzun_kelime)
>>> print  'En uzun kelimenin uzunlugu =', uzunluk  # , ile karakter dizileri peşpeşe yazdırılabilir.
En uzun kelimenin uzunlugu = 70


>>> sonsuz = float('inf')    # float('inf') ile sonsuz değeri elde edilebilir.
>>> 5 < sonsuz                # en küçük sayıyı bulma gibi algoritmalarda kolaylık sağlar.
True
>>> 3 > sonsuz
False
```

python™

Değişkenler

- Değişken tanımlarken tip belirtmeye gerek yoktur.

```
>>> x = 5
>>> type(x)      # değişkenlerin tipi bilinmek isteniyorsa type fonksiyonuyla görülebilir.
<type 'int'>
>>> y = '5'
>>> type(y)
<type 'str'>
>>> z = int(y)    # tip dönüşümü
>>> type(z)
<type 'int'>
```

python™

Kullanıcıyla İletişim

- Dışarıdan girdi almak için:

```
>>> e_posta = raw_input('E-posta adresinizi giriniz:\n')
E-posta adresinizi giriniz:  # burada kullanıcıdan girdi değeri alınır.
>>> print  e_posta
enes@enesates.com
>>> type(e_posta)
<type 'str'>
```

python™

Kullanıcıyla İletişim

```
>>> yas = raw_input('Yasinizi giriniz:\n')
Yasinizi giriniz:
>>> type(yas)
<type 'str'>


>>> yas = int(raw_input('Yasinizi giriniz:\n'))
Yasinizi giriniz:
>>> type(yas)
<type 'int'>
```

python™

.py

- Python programlama diline özgü dosya uzantısıdır.
- Yazılan kodları saklamak için Vim, Geany, PyDev gibi metin düzenleyicileri ve geliştirme ortamlarında

  modül_adı.py

  şeklinde dosyalar (modüller) oluşturulur.

- Kod parçalarının bulunduğu bu dosyaları çalıştırmak için uçbirimde (terminal) aşağıdaki komut verilir.

  >>> python modül_adı.py

- Türkçe karakter sorununu çözmek için dosyanın en üstüne aşağıdaki satırlar eklenmelidir.
  - GNU/Linux için: ***# -*- coding: utf-8 -*-*** Windows için: ***# -*- coding: cp1254 -*-***

python™

Vim

- Uçbirimde çalışabilen, özgür bir metin düzenleyicisidir.
- ***vim dosya_adı.py*** komutu *dosya_adı.py* dosyasını açar, yoksa oluşturur.

- Dosya açıldığında görüntüleme modunda açılmış olur.
  - ***i*** tuşu ile yazma moduna geçilir.
  - ***a*** tuşu imleçten bir sonraki konumdan itibaren yazmaya başlamayı sağlar.
  - ***o*** tuşu bir alt satırdan itibaren yazmaya başlamayı sağlar.
  - ***Esc*** ile yazma modundan çıkılır ve dosya görüntüleme moduna dönülür.
  - ***h, j, k, l*** ya da ok tuşları dosyada yukarı, aşağı, sağa, sola dolaşmayı sağlar.

python™

Vim

- Dosya görüntüleme modunda;
  - ***d*** (sil)
    - ***dd*** (satırı sil)
    - ***7dd*** (7 satır sil)
  - ***y*** (kopyala)
    - ***yy*** (satırı kopya)
    - ***4yy*** (4 satır kopyala)
  - ***p*** (yapıştır)
  - ***u*** (geri al)
  - ***Ctrl + r*** (ileri al)

python™

Vim

- Dosya görüntüleme modunda;
  - ***:w*** (dosyayı kaydet)
  - ***:q*** (vim'den çık)
  - ***:q!*** (kaydetmeden çık)
  - ***:wq*** (kaydet ve çık)

  - ***:set number*** (satır numaralarını göster)
  - ***:10*** (10. satıra git)

  - ***/kelime*** (dosyada girilen kelimeyi ara)
  - ***n*** (kelime birden fazla konumda bulunduysa aralarında dolaşmayı sağlar)

python™

Kontrol Yapıları

if - elif - else
while
for
for - range()
break - continue - else
Fonksiyonlar
Fonksiyonlara Değer Geçirme

python™

if - elif - else

```
yas = int(raw_input('Yaşınızı giriniz: '))
if  yas <= 0:
        print 'Uygun değer girilmedi!'
elif  yas < 18:
        print 'Yaşınıza uygun olmayan, yasaklı sitelere erişmeye çalışıyorsunuz!'
else:
        print 'Bu siteye erişim mahkeme kararıyla engellenmiştir!'
```

- Girintilere dikkat!!
- Girinti için 1 tab (4 boşluk) ya da 2 tab (8 boşluk) bırakılmalıdır.

python™

while

```
cevap = raw_input('Python mu Java mı? ')
while cevap != 'Python':
    print 'Yanlış cevap!'
    cevap = raw_input('Python mu Java mı? ')
```

- Python'da kod blokları **:** karakteri ile başlatılır
- Kod blokları süslü parantezlerle değil, girintilerle kontrol edilir.

python™

for

```
diller = ['Türkçe', 'İngilizce', 'Fransızca', 'Japonca']
for  dil  in  diller:          # diller dizisindeki/listesindeki elemanlar sırayla dil değişkenine atanır.
        print  dil
```

```
for  harf  in  'Python':  # Python karakter dizisindeki karakterler sırayla harf değişkenine atanır.
        print  harf,          #  ,  karakteri alt satıra geçmeyi engeller ve çıktıları yan yana yazar.
```

- Diğer bazı dillerdeki *for each* yapısına benzer.
- Kendisine verilen bir dizi elemanı tek tek dolaşıp işlemeye çalışır.

python™

for - range()

>>> range(10) # 0'dan verilen sayıya kadar (sayı dahil değil) tam sayı dizisi oluşturur.

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(5, 10) # başlangıç ve bitiş değeri alan şekli

[5, 6, 7, 8, 9]

>>> range(0, 10, 3) # başlangıç ve bitiş değerinin yanında artış miktarını da alır.

[0, 3, 6, 9]

```
for  i  in  range(1, 100, 4):   # 1'den 100'e kadar dörder artarak tüm sayıları dolaşır.
        if   i % 5 == 0:          # 5'in katı olan sayıları yazdırır.
                print i,
```

python™

break - continue - else

- Asal sayıları bulan algoritma:

```
for n in range(2, 15):
    for x in range(2, n):
        if n % x == 0:          # kontrol edilen sayının tam böleni var mı kontrolü
            print n, '=', x, '*', n/x
            break               # sayının tam böleni varsa asal sayı değildir ve döngü sonlanır.
    else:    # hiçbir tam böleni bulunmadıysa sayı asaldır.
        print n, 'bir asal sayıdır.'
```

- ***break*** deyimi döngü işleyişini tamamen sonlandırır.
- ***else*** deyimi for döngüsü ile kullanılabilir. for döngüsünden bir hata fırlatılmadıysa (break deyimine denk gelmek gibi) else koşulu çalışır.

python™

break - continue - else

- Çift sayıları bulan algoritma:

```
for sayi in range(2, 10):
    if sayi % 2 == 0:      # sayı çift mi diye kontrol edilir.
        print 'Sıradaki sayı çifttir:', sayi
        continue           # sıradaki sayı çifttir ve döngüdeki mevcut adım sona erer.
    print 'Sıradaki sayı:', sayi
```

- ***continue*** deyimi döngünün tamamını değil, mevcut adımı sonlandırır. Döngü kalan satırları çalıştırmadan bir sonraki adıma atlar.

python™

Fonksiyonlar

- Fibonacci Serileri:

```
def  fib(n):   # fonksiyon tanımı
        a, b = 0, 1
        while  a < n:
                print a,
                a, b = b, a + b
fib(2000)   # fonksiyon çağrımı
```

- ***def*** etiketi, fonksiyon adı ve gerekli parametrelerle fonksiyon tanımlanır.
- Fonksiyon adı ve argümanlarla birlikte fonksiyon çağrımı yapılır.

python™

Fonksiyonlar

```
def  yas_kontrolu(alt_limit = 0, ust_limit = 100):  # varsayılan parametre değerleri
        yas = int(raw_input('Yaşınızı giriniz:\n'))
        if   alt_limit < yas < ust_limit:    return yas
        else:    print  'Aralık dışı değer girdiniz!'

kullanici_yasi = yas_kontrolu(ust_limit = 60)   # varsayılan parametrelere yeni değer atanabilir
if  kullanici_yasi:    print 'Yasiniz:', kullanici_yasi   # dönen değer None değilse
```

- Fonksiyon parametrelerine varsayılan değer atanabilir.
- ***return*** etiketiyle değer döndürülebilir. Varsayılan olarak ***None*** değeri döner.

python™

Fonksiyonlar

```
a = 10
def  fonk():
        global  a      #  fonksiyonun dışında bulunan  a  değişkenini kullanmak için
        a  =  5
        return  a
print  'Fonksiyon dışında a =', a
print  'Fonksiyon içinde a =', fonk()
print  'Fonksiyon dışında a =', a    # fonksiyon içerisinde global değişken a'nın değeri değişmiştir
```

```
def  fonk(a):   #  fonksiyonun içeriğini daha sonra yazmak istediğimizde hata almamak için
        pass      #  bir şey yapmadan geç anlamına gelen  pass  etiketini kullanırız
```

python™

Fonksiyonlara Değer Geçirme

- Fonksiyonlara değişkenlerin kopyaları geçer/aktarılır.

```
>>> x = 5
>>> def  degistir(a):   # yollanan argümanın içeriğini değiştiriyoruz.
. . .          a = 6
>>> degistir(x)
>>> print  x
5                       # değişkenin  kopyası  gittiğinden içeriği değişmedi.
```

python™

Fonksiyonlara Değer Geçirme

- Fonksiyonlara liste gibi yapıların referansları geçer/aktarılır.

```
>>> def  sirala(liste):        # verilen listeyi sıralar.
. . .          liste.sort()
>>> sayilar  =  [4, 2, 1, 10, 6]
>>> sirala(sayilar)            # sayılar  listesinin  referansını  yolladık.
>>> sayilar
[1, 2, 4, 6, 10]               # listenin  referansı  gittiğinden dolayı içeriği değişti.
```

python™

Fonksiyonlara Değer Geçirme

- Fonksiyonlara listelerin kopyalarını geçirme:

```
>>> def  sirala(liste):
. . .          liste.sort()
>>> sayilar  =  [4, 2, 1, 10, 6]
>>> sirala( list(sayilar) )          # listenin bir  kopyasını  oluşturup onu yolluyoruz.
>>> sayilar
[4, 2, 1, 10, 6]                     # listenin  kopyası  gittiğinden içeriği değişmedi.
```

python™

Veri Yapıları

Listeler (Lists)
Yığıtlar (Stacks)
Kuyruklar (Queues)
filter(), map(), reduce()
Demetler (Tuples)
Kümeler (Sets)
Sözlükler (Dictionaries)

python™

Listeler (Lists)

- Diğer dillerdeki dizilere benzetebiliriz.
- Elemanları farklı tiplerde (tam sayı, karakter dizisi, liste vb.) olabilir.

```
>>> dagitim = ['Trisquel', 'Fedora', 'Ubuntu', 'Mint', 'Gentoo']
>>> dagitim          # listeyi yazar.
['Trisquel', 'Fedora', 'Ubuntu', 'Mint', 'Gentoo']
>>> print  dagitim[1]   # listelerde elemanların sırası 0'dan başlar.
Fedora                  # listenin 2. elemanı
>>> len(dagitim)    # listenin boyutu/uzunluğu
5
```

python™

Listeler (Lists)

```
>>> dagitim.append('Debian')        # listeye eleman ekler
['Trisquel', 'Fedora', 'Ubuntu', 'Mint', 'Gentoo', 'Debian']

>>> dagitim.insert(5, 'Ubuntu')     # listenin girilen sırasına elemanı ekler
['Trisquel', 'Fedora', 'Ubuntu', 'Mint', 'Gentoo', 'Ubuntu', 'Debian']

>>> dagitim.remove('Ubuntu')        # listeden ilk Ubuntu değerini siler
['Trisquel', 'Fedora', 'Mint', 'Gentoo', 'Ubuntu', 'Debian']

>>> dagitim.pop(4)                  # listeden verilen sıradaki elemanı silip, değerini döndürür
'Ubuntu'
```

python™

Listeler (Lists)

```
>>> sayilar  =  [ ]      # boş liste
[ ]
>>> sayilar.append(42)
[42]
>>> sayilar.insert(0, 54.9)
[54.9, 42]
```

```
>>> dagitim.extend(sayilar) ya da  dagitim += sayilar  # sayilar listesini dagitim listesine ekler.
['Trisquel', 'Fedora', 'Mint', 'Gentoo', 'Debian', 54.9, 42]
```

python™

Listeler (Lists)

```
>>> dagitim.index('Gentoo')    # Gentoo  değerinin sırasını döndürür
3
>>> dagitim[3:6]      # listenin 3 ile 6. sıraları arasındaki elemanlarını döndürür
['Gentoo', 'Debian', 54.9]


>>> dagitim.sort()          # listedeki elemanları alfabetik olarak sıralar
[42, 54.9, 'Debian', 'Fedora', 'Gentoo', 'Mint', 'Trisquel']
>>> dagitim.reverse()     # listedeki elemanların sırasını ters çevirir
['Trisquel', 'Mint', 'Gentoo', 'Fedora', 'Debian', 54.9, 42]
```

python™

Listeler (Lists)

```
>>> dagitim.append('Mint')
['Trisquel', 'Fedora', 'Mint', 'Gentoo', 'Debian', 54.9, 42, 'Mint']

>>> print dagitim.count('Mint'),  dagitim.count('Fedora'),  dagitim.count('Ubuntu')
2, 1, 0            # Mint, Fedora ve Ubuntu  değerlerinin listede kaçar kez geçtiğini bulur

>>> 'Trisquel'  in  dagitim    # Verilen değerin listede bulunup bulunmadığına bakar
True
>>> 'Ubuntu'  in  dagitim
False
```

python™

Listeler (Lists)

```
>>> kare = [ ]
>>> for x in range(10):
. . .     kare.append(x ** 2)     # x sayısının karesini kare listesine ekler.
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> kare = [x ** 2 for x in range(10)]    # üstteki algoritmanın farklı şekilde yazılışı
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> del kare[3:5]     # kare listesindeki 3. elemandan 5. elemana kadar tüm elemanları siler.
[0, 1, 4, 25, 36, 49, 64, 81]
```

python™

Listeler (Lists)

```
>>> matrix = [[1, 2, 3, 4],          # 3 x 4'lük matris (iç içe dizi)
...            [5, 6, 7, 8],
...            [9, 10, 11, 12], ]


>>> transpoze = []
>>> for i in range(4):               # matrisin transpozesini alma yöntemi.
...         transpoze.append([row[i] for row in matrix])
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

python™

Yığıtlar (Stacks)

- Yığıtlar, *son giren ilk çıkar (last-in, first-out)* mantığıyla davranan listelerdir.
- Eklenecek eleman sona eklenir: ***append()***
- Çıkarılacak eleman sondan çıkarılır: ***pop()***

```
>>> yigit  =  [3, 2, 8]
>>> yigit.append(6); yigit.append(9)
[3, 2, 8, 6, 9]
>>> yigit.pop()
9
>>> yigit
[3, 2, 8, 6]
```

python™

Kuyruklar (Queues)

- Kuyruklar, ilk *giren ilk çıkar (first-in, first-out)* mantığıyla davranan listelerdir. Banka kuyruklarına benzetebiliriz.
- Etkin çalışması için ***collections.deque*** kulanırız.

```
>>> from collections import deque
>>> kuyruk = deque(['Kemal', 'Ali', 'Merve'])
>>> kuyruk.append('Zeytin')    # Zeytin kuyruğa girer.
>>> kuyruk.popleft()    # İlk sıradaki kişi kuyruktan çıkar.
'Kemal'
>>> kuyruk
deque(['Ali', 'Merve', 'Zeytin'])
```

python™

filter(), map(), reduce()

```
>>> def  f(x):   return x%2 != 0  and  x%3 != 0      # sayı 2 ve 3'e bölünmüyorsa True değeri döner.
>>> filter(f,  range(2, 25))      # f fonksiyonuna  2'den 25'e kadar tüm tam sayıları sırayla yollar.
[5, 7, 11, 13, 17, 19, 23]        # f fonksiyonunun sonucunu True yapan tüm sayıların listesi döner.

>>> def  kup(x):  return  x*x*x      # sayının küpünü döndürür.
>>> map(kup,  range(1, 11))       # kup fonksiyonuna 1'den 11'e kadar tüm tam sayıları sırayla yollar.
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]   # yollanan sayıların küplerinin listesi elde edilir.

>>> def  ekle(x, y):  return  x+y   # sayıları birbirine ekler.
>>> reduce(ekle,  range(1, 11)    # ekle fonksiyonuna 1'den 11'e kadar tüm tam sayıları sırayla yollar.
55              # ekle  fonksiyonunun parametreleri yollanan sayı ve bir önceki fonksiyonun sonucudur.
```

python™

Demetler (Tuples)

- Listelere benzer.
- Listelerdeki gibi elemanlar üzerinde değişiklik yapamayız.
  - Değişikliğe izin vermediği için listelerden daha güvenilirdir.
  - Kazara değiştirmek istemediğimiz verileri içeren liste kullanmak istiyorsak kullanabiliriz.
- Listelere göre daha hızlı çalışırlar.
  - Uygulama içerisinde sonradan değiştirmeyeceğimiz verileri gruplamak için kullanabiliriz.

```
>>> demet = ()     #  boş demet
>>> demet  =  'GNU', 'Linux', 60, 70   ya  da
>>> demet  =  ('GNU', 'Linux', 60, 70)
```

python™

Demetler (Tuples)

```
>>> demet  =  ('GNU',)    # tek elemanlı demette elemanın yanına  ,  koymamız gerekir.
>>> demet2  =  ('GNU')   # ,  koymazsak girdiyi karakter dizisi olarak algılar.
>>> print  type(demet),  type(demet2)
<type 'tuple'> <type 'str'>
```

- Demet oluşturma işlemine ***demetleme (packing***), tersine ***demet çözme (unpacking)*** denir.

```
>>> aile  =  'Anne', 'Baba', 'Kardes'    # demetleme (packing)
>>> a, b, c  =  aile       # demet çözme(unpacking)
>>> print  a, b, c
Anne Baba Kardes
```

python™

Kümeler (Sets)

- Matematikteki kümelere benzer:
  - Her elemanı benzersizdir (tekil).
  - Sıra önemsizdir, sadece elemanlar bir arada toplanmıştır.
  - Kesişim, birleşim, fark gibi işlemlerden geçirilebilir.
- Üyelik kontrollerinde ve veri tekrarını önleme işlemlerinde kullanılabilir.

```
>>> sepet = ['elma', 'portakal', 'elma', 'armut', 'portakal', 'muz']     # sepet listesi
>>> meyve = set(sepet)          # listeden küme oluşturma
>>> meyve
set(['elma', 'armut', 'portakal', 'muz'])      # tekrar eden elemanlar teke inmiştir.
```

python™

Kümeler (Sets)

```
>>> 'armut' in meyve      # armut meyve kümesinin elemanı mı kontrolü
True
>>> 'lahana' in meyve
False


>>> kume = set('adana')   # adana karakter dizisinin kümesi
>>> for i in kume:
...         print i,
a d n              # tekrar eden a karakteri teke inmiştir.
```

python™

Kümeler (Sets)

```
>>> yedek = meyve.copy()   # meyve kümesinin tüm içeriğini yedek kümesine kopyalar.
>>> meyve.add('kiraz')     # meyve kümesine kiraz elemanını ekler.
set(['elma', 'kiraz', 'armut', 'portakal', 'muz'])

>>> meyve.discard('armut')   # meyve kümesinden armut elemanını çıkarır.
set(['elma', 'kiraz', 'portakal', 'muz'])
>>> meyve.remove('elma')     # kümeden elma elemanını çıkarır, eleman yoksa hata verir.
set(['kiraz', 'portakal', 'muz'])
>>> meyve.clear()            # meyve kümesinin tüm içeriğini temizler.
set([ ])
```

python™

Kümeler (Sets)

```
>>> k1 = set([1, 2, 3]); k2 = set([3, 10, 2, 1])
>>> k2 | k1        # birleşim
set([1, 2, 3, 10])
>>> k2 & k1        # kesişim
set([1, 2, 3])


>>> k2 - k1        # fark
set([10])
>>> k2 >= k1       # kapsama
True
```

python™

Sözlükler (Dictionaries)

- ***Anahtar:Değer*** ikililerinden oluşan veri yapısıdır.

```
>>> telefon  =  {'Samet': 2390,  'Can': 5123, 'Nalan': 8273}
>>> telefon['Ayla']  = 4732  # sözlüğe (telefon rehberi) Ayla anahtarını ve 4732 değerini ekler.
>>> telefon
{'Nalan': 8273, 'Ayla': 4732, 'Can': 5123, 'Samet': 2390}

>>> telefon['Can']       # Can'ın telefon numarasını bulur.
5123
>>> telefon['Nalan']  =  3182     # Nalan'ın telefon numarasını değiştirir.
```

python™

Sözlükler (Dictionaries)

```
>>> telefon.keys()        # sözlükteki anahtarları listeler.
['Nalan', 'Ayla', 'Can']
>>> telefon.values()     # sözlükteki değerleri listeler.
[3182, 4732, 5123]
>>> 'Ayla'  in  telefon   # sözlükte  Aylin  anahtarı var mı diye kontrol eder.
True


>>> del  telefon['Samet']    # Samet  anahtarını sözlükten değeriyle birlikte siler.
{'Nalan': 3182, 'Ayla': 4732, 'Can': 5123}
>>> telefon.clear()        # sözlüğün içeriğini tamamen temizler.
{ }
```

python™

Modüller

Modül Nedir?
Modülleri İçe Aktarmak
os Modülü
math Modülü
random Modülü
datetime Modülü
urllib2 Modülü

python™

Modül Nedir?

- İçerisinde
  - Karakter dizileri, sayılar
  - Değişkenler
  - Listeler, sözlükler, demetler
  - Fonksiyonlar

  gibi kod parçacıkları bulunan dosyalardır.
- .py uzantısına sahiptir.
- Betik (script) olarak da bilinir.
- Kısaca yazılan Python uygulamaları birer modüldür.
- ***python modül_adı.py*** şeklinde çağrılabilir.

python™

Modülleri İçe Aktarmak

- hesap_makinesi.py adında bir modül yaratalım:

```
def carp(liste):
    a = 1
    for i in liste:
        a = a * i
    print a
```

```
>>> import hesap_makinesi      # hesap_makinesi modülünü içe aktardık (import ettik).
>>> sayilar = [45, 66, 76, 12]
>>> hesap_makinesi.carp(sayilar)
2708640                # hesap_makinesi modülünün carp fonksiyonuna sayılar listemizi yolladık.
```

python™

Modülleri İçe Aktarmak

- hesap_makinesi modülümüzü değiştirelim:

```
def carp(liste):
    a = 1
    for i in liste:
        a = a * i
    return a
```

```
>>> reload(hesap_makinesi)        # değişen modülü tekrar içe aktarma
>>> sonuc = hesap_makinesi.carp(sayilar)
>>> print sonuc
2708640
```

Modülleri İçe Aktarmak

- ***import*** *modül_adı:*
  - Bir modülü tüm içeriğiyle birlikte içe aktarır.
- ***from*** *modül_adı* ***import*** ***
  - Modüldeki "_" ile başlayanlar hariç tüm içeriği içe aktarır.
- ***from*** *modül_adı* ***import*** *fonksiyon_adı, değişken_adı, ...*
  - Modüldeki istediğimiz fonksiyonları ve değişkenleri (referansı değil kopyası) içe aktarır.
- ***import*** *modül_adı* ***as*** *yeni_ad*
  - Modülün adını değiştirerek kullanmamızı sağlar. Uzun modül adlarını kısaltabiliriz.
- Modül farklı dizindeyse adresini vermek ya da yola (PATH) eklemek gerekir.

```
>>> dir(modül_adı)      # modül içindeki fonksiyon ve nitelikleri (değişken vb.) listeler.
```

python™

Modülleri İçe Aktarmak

```
>>> import  hesap_makinesi
>>> hesap_makinesi.carp(sayilar)

>>> import  hesap_makinesi  as  hsp
>>> hsp.carp(sayilar)

>>> from  hesap_makinesi  import *
>>> carp(sayilar)   # modül adını belirtmediğimizden aynı isimde farklı bir fonksiyon yazmamalıyız.

>>> from  hesap_makinesi  import carp
>>> carp(sayilar)
```

python™

os Modülü

- Kullanılan işletim sistemiyle ilgili işlemler yapma olanağı sunar.
- Uygulamaların farklı işletim sistemleri üzerinde çalışmasını kolaylaştırır.

```
>>> import os                    # os modülünü içe aktarır.
>>> for icerik in dir(os):       # os modülünün tüm içeriğini (fonksiyon, değişken vs.) listeler.
. . .        print icerik,
```

… makedirs minor mkdir mkfifo mknod name nice open openpty pardir path pathconf pathconf_names pathsep pipe popen popen2 popen3 popen4 putenv read readlink remove removedirs rename renames rmdir …

python™

os Modülü

- ***name*** niteliği:
  - İşletim sistemi hakkında bilgi verir.
  - Uygulamanın işletim sistemine göre işlem yapması sağlanabilir.

```
>>> os.name
'posix'    # GNU/Linux: posix;    Windows: nt, dos, ce;    Macintosh: mac
```

- ***getcwd*** fonksiyonu:
  - O anda hangi dizin içinde bulunduğumuzu gösterir.

```
>>> os.getcwd()
'home/enes'
```

python™

os Modülü

- ***listdir*** fonksiyonu:
  - Dizin içerisindeki dosya ve dizinleri listeler.

```
>>> dosya_dizin = os.listdir('/home/enes')
>>> for a in dosya_dizin: print a,
… .gnash Public Pictures .gimp-2.6 Downloads Music Desktop .gconf Documents …
```

- ***chdir*** fonksiyonu:
  - Bulunduğumuz dizini değiştirir.

```
>>> os.chdir(os.pardir)    # pardir (parent directory) ile bir üst dizini elde ederiz.
>>> print os.getcwd()    # mevcut dizin
/home
```

python™

os Modülü

- ***mkdir*** ve ***makedirs*** fonksiyonları:
  - Dizin veya dizinler oluştururlar.

```
>>> os.mkdir('test')      # dizin oluşturur.
>>> os.makedirs('python_kodlar/moduller/os_modulu')    # iç içe dizinler oluşturur.
```

- ***rmdir*** ve ***removedirs*** fonksiyonları:
  - İçi boş dizin veya dizinleri siler.

```
>>> os.rmdir('test')      # dizini siler.
>>> os.removedirs('python_kodlar/moduller/os_modulu')   # iç içe dizinleri siler.
```

python™

os Modülü

- ***remove*** fonksiyonuyla mevcut bir dosya silinebilir:

```
>>> os.remove('deneme.txt')
```

- ***system*** fonksiyonuyla sistem fonksiyonları çalıştırılabilir:

```
>>> os.system('ls')        # sistemdeki ls komutunu çalıştırır, dizinin içeriğini listeler.

Documents  hesap_makinesi.py   Music   Public     Videos deneme.pyc  Desktop   Downloads
hesap_makinesi.pyc  Pictures  Templates  workspace


>>> os.system('cat  hesap_makinesi.py')  # dosya içeriğini listeler.

def carp(liste): …
```

python™

math Modülü

```
>>> import  math

>>> dir(math)                # math  modülünün içerisindeki tüm özellikleri listeler.
... 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', ...

>>> math.log(1024, 2)   # 1024'ün 2 tabanına göre logaritması
10.0

>>> math.sqrt(81)        # 81'in karekökü
9
```

python™

random Modülü

```
>>> import  random ;  dir(random)   # random  modülünü içe aktarıp içeriğini listeledik.

>>> random.random()          # 0.0, 1.0 aralığında ondalıklı sayı seçer.
0.060059391318312616

>>> random.uniform(1, 10)   # 1.0, 10.0 (dahil değil) aralığında ondalıklı sayı seçer.
5.0059275809358095

>>> random.randint(1, 10)    # 1, 10 aralığında (sınır değerler dahil) bir tam sayı seçer.
6
```

random Modülü

```
>>> random.choice(['elma', 'armut', 'zeytin'])   # listeden gelişigüzel bir elemanı seçer.
'zeytin'


>>> sayilar  =  [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(sayilar)       # listedeki elemanların sırasını karıştırır.
>>> sayilar
[2, 5, 7, 6, 4, 1, 3]


>>> random.sample([1, 2, 3, 4, 5],  3)   # listeden gelişigüzel 3 eleman seçer.
[4, 3, 5]
```

python™

datetime Modülü

```
>>> from datetime import date    # datetime modülünden date fonksiyonunu içe aktarır.

>>> bugun = date.today()          # bugünün tarih bilgisini alır.
>>> bugun.strftime('%d-%m-%y')   # düzgün şekilde görüntülemek için
'23-03-13'

>>> dogum_gunu = date(1967, 3, 29)   # tarih bilgisi elle verilebilir.
>>> yas = bugun - dogum_gunu         # iki tarih arasındaki fark
>>> yas.days
16796
```

python™

urllib2 Modülü

- İnternet ile veri iletişimi sağlar.

```
>>> import  urllib2
```

```
>>> cevap = urllib2.urlopen('http://python.org/')   # web adresine bağlanma
>>> html = cevap.read()   # HTML kaynağını dosya okur gibi okuma
>>> print html
```

python™

Dosya İşleme

Dosya Açma ve Oluşturma
Dosyaya Yazma
Dosyadan Okuma
Dosya Konum Belirteci
Dosyaya Değişken Yazma
Dosya Silme
Belirli Konuma Satır Ekleme

python™

Dosya Açma ve Oluşturma

- Bir dosyayı farklı modlarda açabiliriz.
- **r** (read):
  - Dosyayı okuma modunda açar.
  - Dosya yoksa hata verir.
- **w** (write):
  - Dosyayı yazma modunda açar.
  - Dosya yoksa oluşturur, varsa eski tüm içeriği siler.
- **a** (append):
  - Dosyayı yazma modunda açar.
  - Dosya yoksa oluşturur, varsa eski içeriğe ekleme yapar.

python™

Dosyaya Yazma

```
>>> dosya  =  open('deneme.txt', 'w')              # dosyayı yazma modunda açtık.
>>> dosya.write('Python özgür bir yazılımdır.')  # dosyaya karakter dizisini yazdırdık.
>>> dosya.close()      # dosyayı kapattık.


>>> dosya  =  open('deneme.txt', 'a')     # dosyayı ekleme modunda açtık.
>>> dosya.write('\nAdını Monty Python\'s Flying Circus adlı komedi dizisinden almıştır.')
>>> dosya.writelines(['\nBasit söz dizilimlidir.',  '\nGirintileri sever.',
                                                  '\nOkunabilirliğe önem verir.'])
      # bir listenin içeriğini (karakter dizilerinden oluşan) dosyaya yazdırdık.
>>> dosya.close()
```

python™

Dosyadan Okuma

```
>>> dosya  =  open('deneme.txt', 'r')   # dosyayı okuma modunda açtık


>>> dosya.read()          # dosyayı ham (Türkçe karaktersiz, \n gibi çıktılarla) haliyle okuduk.
"Python \xc3\xb6zg\xc3\xbcr bir yaz\xc4\xb1l\xc4\xb1md\xc4\xb1r.\ ...
>>> print  dosya.read()  # dosyayı düzgün bir çıktıyla okuduk.
Python özgür bir yazılımdır. …


>>> print  dosya.readline()    # dosyadan tek satır okuduk.
>>> print  dosya.readlines()  # çıktının bir liste olacağı şekilde dosyadan tüm satırları okuduk.
```

python™

Dosya Konum Belirteci

- Dosya konum belirteci zamanla farklı konumlara (dosya sonu vb.) taşınır.
- Tekrar okuma ya da yazma yapacağımız zaman konum belirtecini istediğimiz noktaya taşıyabiliriz.

```
>>> dosya.seek(0)      # dosya konum belirtecini dosya başına taşır.
>>> dosya.seek(10)     # dosya konum belirtecini 10. karakterin başına getirir.
>>> dosya.seek(5, 0)   # belirteci dosya başından itibaren 5 birim kaydırır.
>>> dosya.seek(5, 1)   # belirteci bulunulan konumdan itibaren 5 birim kaydırır.
>>> dosya.seek(-5, 2)  # belirteci dosyanın sonundan başa doğru 5 birim kaydırır.
>>> dosya.tell()       # belirtecin konumunu verir.
```

python™

Dosyaya Değişken Yazma

```
>>> x  =  50


>>> dosya  =  open('deneme.txt', 'a')
>>> dosya.write(x)   # dosyaya değişken yazabiliriz.
… TypeError …       # karakter dizisi tipinde olmayan bir değişken yazdırmak tip hatası verir.


>>> dosya. write(str(x))  # yazacağımız değişkeni karakter dizisine çevirerek yazdırırız.
>>> dosya.close()
```

python™

Dosya Silme

```
>>> import  os

>>> os.remove('deneme.txt')
```

python™

Belirli Konuma Satır Ekleme

```
>>> kaynak = open('deneme.txt')   # dosyayı okumak için açtık.
>>> hedef = open('test.txt', 'w')   # hedef bir dosya oluşturduk.
>>> oku = kaynak.readlines()   # kaynak dosyadan tüm satırları okuduk.

>>> for satirlar in oku[:2]:
. . .        hedef.write(satirlar)   # kaynak dosyadaki ilk 2 satırı hedef dosyaya yazdık.

>>> hedef.write('Guido van Rossum tarafından geliştirilmeye başlanmıştır.\n')  # yeni satır
>>> for satirlar in oku[2:]: hedef.write(satırlar)   # kalan satırları hedef dosyaya yazdık.
>>> kaynak.close(); hedef.close()
```

python™

Hata Yakalama

try - except

python™

try - except

- ***try , except*** etiketleri ile olası hatalar kontrol altına alınabilir:
  - try:

```
        bolunen =  int(raw_input("Bölme işlemi için bölünecek sayıyı girin: "))
        bolen     =  int(raw_input("Şimdi de bölecek sayıyı girin: "))
        bolum    =  float(bolunen) / bolen
        print bolum
    except ZeroDivisionError:  # kullanıcı bölen sayı için 0 değeri girerse bölmede hata fırlayacaktır.
        print  'Lütfen sayıyı 0\'a bölmeye çalışmayın!'   # hata fırladığında düzgün hata mesajı verilebilir.
    except ValueError:          # kullanıcının uygun değer girmemesi hatası da yakalanabilir.
        print  'Lütfen harf değil, sayı girin!'
```

  - except (ZeroDivisionError, ValueError):

```
        pass        # Hata varsa hiçbir şey yapma
```

python™

Sınıflar

Sınıf Tanımlama
Nesne Nedir?
Nesne Yaratma
Nesne Özelliği (Attribute)
self
Kalıtım (Miras Alma)
Boş Sınıf ve Özel (Private) Değişken

python™

Sınıf Tanımlama

- Python'da sınıf yapısı ***class*** etiketi ve ***sınıfın adı*** ile tanımlanır.
- Sınıfın içinde sınıfa özgü değişkenler, fonksiyonlar vb. özellikler bulunur.

- Penguen adında, isim ve memleket özellikleri bulunan basit bir sınıf:

```
class Penguen:
    def __init__(self, penguen_ismi, yasadigi_yer):    # varsayılan değerlerle sınıf oluşturur.
        self.isim = penguen_ismi
        self.memleket = yasadigi_yer
        self.ruh_hali = 'çok aç'
```

python™

Nesne Nedir?

- Bir sınıftan yaratılmış elemandır.
- Özellikler (değişkenler vb.) ve yeteneklerden (fonksiyonlar) oluşur.

- Python'da her şey nesnedir.

```
>>> dir(5)
```

```
>>> x = 'deneme'
>>> dir(x)
>>> x.upper()
'DENEME'
```

python™

Nesne Yaratma

- ***penguen.py*** adında bir modül oluşturup içerisine sınıfımızı ekleyelim:

```
class Penguen:
    def __init__(self, penguen_ismi, yasadigi_yer):
        self.isim = penguen_ismi
        self.memleket = yasadigi_yer
        self.ruh_hali = 'çok aç'
```

 # -- coding: utf-8 -*-*

```
>>> import penguen      # penguen modülünü içe aktardık.
>>> pingu = penguen.Penguen('Pingu', 'Kuzey Kutbu')
    # penguen modülünün Penguen sınıfından pingu isminde nesne yaratmış olduk.
```

python™

Nesne Özelliği (Attribute)

```
>>> print  pingu.ruh_hali    # pingu  nesnesinin  ruh_hali  özelliğine eriştik.
çok aç

>>> print  pingu.isim, '-', pingu.memleket
Pingu – Kuzey Kutbu

>>> del  pingu.ruh_hali     # pingu  nesnesinin  ruh_hali  özelliğini yok ettik.
>>> pingu.ruh_hali          # pingu  nesnesinin olmayan bir özelliğine erişmeye çalışıyoruz.
pingu instance has no attribute 'ruh_hali'
```

python™

Nesne Özelliği (Attribute)

- ***penguen.py*** modülümüzü güncelleyelim:

```
class  Penguen(object):          # Yeni tip sınıf tanımlama (object sınıfını miras almış)

    def  __init__(self,  penguen_ismi, yasadigi_yer):
                                                                    # -*- coding: utf-8 -*-
        self.isim  =  penguen_ismi
        self.memleket  =  yasadigi_yer
        self.ruh_hali  =  'çok aç'
        self.balik_deposu  =  40          # Penguen sınıfına  balik_deposu  özelliği ekledik.
    def  karnini_doyur(self):    # Penguen sınıfına  karnını_doyur  fonksiyonu/özelliği ekledik.
        self.ruh_hali  =  'karnı tok'
        self.balik_deposu  -=  30
    def  balik_tut(self):                         # Penguen sınıfına  balık_tut  fonksiyonu/özelliği ekledik.
        self.balik_deposu  +=  80
```

python™

self

- Fonksiyonlardaki ***global*** ifadesine benzer.
  - Sınıf genelinde geçerli olacak/sınıfa özgü değişken için, değişkene bu sınıfın bir özelliğidir anlamını yükler.

- ***pingu*** örneği için, self ifadesini pingu nesnesinin yerini tutan bir kelime olarak zihnimizde canlandırabiliriz.

- Bir sınıftaki tüm fonksiyonların ilk parametresi self olmalıdır.

python™

Kalıtım (Miras Alma)

- İmparator Penguen sınıfı yaratmak istersek:
  - Bu sınıf Penguen sınıfıyla benzer olacaktır.
  - Sonuçta bir penguenle, imparator penguenin ortak birçok özelliği vardır.

- İmparator Penguen sınıfı için Penguen sınıfındaki kodların tamamını tekrar yazmamız gerekmez.

- Python'daki kalıtım özelliğini kullanarak İmparator Penguen sınıfını Penguen sınıfından türetebiliriz.

python™

Kalıtım (Miras Alma)

- ***penguen.py*** modülümüze ekleme yapalım:

```
class ImparatorPenguen(Penguen):   # Penguen sınıfından miras aldık.
    def __init__(self, penguen_ismi, yasadigi_yer):
        Penguen.__init__(self, penguen_ismi, yasadigi_yer)  # Penguen'deki init'i kopyalar.
        del self.ruh_hali            # Her zaman tok olduğundan ruh hali özelliğine gerek yok.
        self.balik_deposu = 600  # balik_deposu özelliğini imparator için güncelliyor.
    def balik_tut(self):             # balik_tut fonksiyonunu imparator için güncelliyor.
        self.balik_deposu += 210
```

-*- coding: utf-8 -*-

- Penguen sınıfından değişikliğe uğrayacak özelliklerin ***üstüne yazılabilir***.
- Artık gerek olmayan özelliklere ise ***del*** etiketi ile erişim engellenebilir.

python™

Boş Sınıf ve Özel (Private) Değişken

- **pass** etiketiyle içeriğini daha sonra dolduracağımız sınıf oluşturabiliyoruz:

```
>>> class  Kitap(object):    pass          # boş bir sınıf oluşturduk.
>>> vakif  =  Kitap()        # vakif  adında bir  Kitap  nesnesi oluşturduk.
>>> type(vakif)              # vakif  nesnesinin tipine baktık.
<class '__main__.Kitap'>

>>> vakif.isim  =  'Vakıf'    # vakif  nesnesine özellikler ekleyip, değerlerini atayabiliyoruz.
>>> vakif.yazar  =  'Isaac Asimov'
```

- Sınıfın özelliklerine sınıf dışından erişimi kısıtlamak için __ etiketini kullanabiliriz:

```
>>> vakif.__fiyat  =  23     # vakif  nesnesine ait  fiyat  özelliğini __  etiketiyle özel yaptık.
```

python™

Pokemon Oyunu

pokemon.py

```
#-*-coding:utf-8-*-
import random

class Pokemon(object):
    def __init__(self, isim, can, sahip):
        self.isim = isim
        self.can = can
        self.seviye = 1
        self.sahip = sahip

    def saldir(self,hedef):
        hedef.can -= random.randint(1,10)  # hedefin canını 1,10 arasında bir değer kadar azaltır.

    def evril(self):
        self.seviye += 1
```

python™

Pokemon Oyunu

elektrik_pokemonu.py

```
#-*-coding:utf-8-*-
from pokemon import *
import su_pokemonu as su


class ElektrikPokemonu(Pokemon):
    def __init__(self, isim, can, sahip):
        Pokemon.__init__(self, isim, can, sahip)
        self.ustun = [su.SuPokemonu]

    def carp(self, hedef):
        if type(hedef) in self.ustun:    # hedeften üstün müyüz?
            hedef.can -= random.randint(10,30)
        else:
            self.saldir(hedef)           # hedeften üstün değilsek normal saldırı yapabiliyoruz.
```

python™

Pokemon Oyunu

su_pokemonu.py

```
#-*-coding:utf-8-*-
from pokemon import *
import ates_pokemonu as ates


class SuPokemonu(Pokemon):
    def __init__(self, isim, can, sahip):
        Pokemon.__init__(self, isim, can, sahip)
        self.ustun = [ates.AtesPokemonu]

    def puskurt(self, hedef):
        if type(hedef) in self.ustun:    # hedeften üstün müyüz?
            hedef.can -= random.randint(10,30)
        else:
            self.saldir(hedef)           # hedeften üstün değilsek normal saldırı yapabiliyoruz.
```

python™

Pokemon Oyunu

ates_pokemonu.py

```
#-*-coding:utf-8-*-
from pokemon import *
import elektrik_pokemonu as elk

class AtesPokemonu(Pokemon):
    def __init__(self, isim, can, sahip):
        Pokemon.__init__(self, isim, can, sahip)
        self.ustun = [elk.ElektrikPokemonu]

    def yak(self, hedef):
        if type(hedef) in self.ustun:    # hedeften üstün müyüz?
            hedef.can -= random.randint(10,30)
        else:
            self.saldir(hedef)           # hedeften üstün değilsek normal saldırı yapabiliyoruz.
```

python™

Pokemon Oyunu

gotta_catch_em_all.py

```
#-*-coding:utf-8-*-
import elektrik_pokemonu as elk, su_pokemonu as su, ates_pokemonu as ates

pikachu = elk.ElektrikPokemonu('Pikachu', 100, 'Ash')  # Ash'in Pikachu'su canı 100 olarak yaratıldı.
staryu  = su.SuPokemonu('Staryu', 80, 'Misty')         # Misty'nin Staryu'su canı 80 olarak yaratıldı.
vulpix  = ates.AtesPokemonu('Vulpix', 85, 'Brock')     # Brock'un Vulpix'i canı 85 olarak yaratıldı.

print pikachu.isim, ' -> ', staryu.isim, ': Pika Pika'
pikachu.saldir(staryu)
print 'Pikachu: ', pikachu.can, '\nStaryu:', staryu.can

print '\n', staryu.sahip, ': Seni seçtim ', staryu.isim
staryu.puskurt(pikachu)
print 'Pikachu: ', pikachu.can, '\nStaryu:', staryu.can

print '\n', vulpix.sahip, ': Seni seçtim ', vulpix.isim, ' durdur onları, giiit!!'
vulpix.yak(pikachu)
vulpix.yak(staryu)
print 'Pikachu: ', pikachu.can, '\nStaryu: ', staryu.can, '\nVulpix: ', vulpix.can
```

python™

Kaynaklar

- [1] http://docs.python.org/2/tutorial/index.html
- [2] http://www.istihza.com/py2/icindekiler_python.html
- [3] http://yzgrafik.ege.edu.tr/~tekrei/dersler/bbgd_p/
- [4] http://kodveus.blogspot.com/search?q=python
- [5] http://en.wikipedia.org/wiki/Python_%28programming_language%29
- [6] http://tr.wikipedia.org/wiki/Python_%28programlama_dili%29
- [7] http://www.tuncercolak.net/blog/vim-vi-improved-editoru/
- [8] http://www.python.org/community/logos/
- [9] http://www.gnu.org/graphics/babies/BabyGnuAlpha.png
- [10] http://tux.crystalxp.net/en.id.1871-rap-tux.html

python™

DİNLEDİĞİNİZ İÇİN

TEŞEKKÜRLER...

Nicolas Rougier (c) 2005 http://www.loria.fr/~rougier