PYTHON İLE PROGRAMLAMAYA GİRİŞ

Cihangir BEŞİKTAŞ

cihangir[at]akademi[dot]EnderUNIX[dot]org

http://www.EnderUNIX.org

13 Mayıs 2008

İÇİNDEKİLER:

1.PYTHON DİLİNİN ÖZELLİKLERİ

* Nesneye yönelikdir.
* Özgürdür.
* Yorumlamalı ve derlemelidir. Her ne kadar sadece yorumlamalı gözükse de *"byte code"* biçiminde derleme yapılır.
* Taşınabilir.
* Güçlüdür. İhtiyacınız olabilecek hemen hemen tüm özellikler mevcuttur.
* Hızlıdır.
* Ticari uygulamalar geliştirmeye uygundur. Bunun için kodu bytecode şeklinde verebilirsiniz ya da *"py2c"* ile C modülüne çevirip derleyebilirsiniz.
* Yazılımı kolaydır.
* Öğrenmesi kolaydır.

2.PYTHON' A GİRİŞ

Öncelikle python sisteminize yok ise www.python.org sitesinden sisteminize uygun olanı indirebilirsiniz. Ayrıca python ın yanında bazı modüllere ihtiyacınız olabilir. Eğer UNIX kullanıyorsanız bu modülleri paket yöneticiniz aracılığıyla yükleyebilirsiniz.
Python iki şekilde kullanılabilir. Bunlardan biri etkileşimli python kabuğunu kullanmak. Diğeri ise programları dosyaya yazmaktır. Etkileşimli kabuk daha çok kısa programlar için ve genelde test etmek için kullanılır.

3.ARİTMETİK İŞLEMLER

+,-,*,/ işlemleri mevcuttur. Yalnız bunları kullanırken tip uyuşmazlığına dikkat etmek gerekir. Ayrıca **'**'** üs almak için kullanılır.
Örnek:

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

Karmaşık sayılar üzerinde de işlem yapılabilir:

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

4.VERİ TİPLERİ

Python daki standart veri tipleri *"sayılar(integer, float), sözcükler(string, character), listeler(lists), tüpler(tuples), sözlükler(dictionaries), dosyalar(files)"* dır.

Değişken isimleri rakamla başlayamaz. Tanımlarken türünü belirtmenize gerek yoktur. Otomatik olarak algılanır.

4.1.Sayı değişkenleri:

```
>>>a=6
>>>b=7
>>>a+b
13
>>>c=(1+1j)
```

```
>>>d=(2+3j)
>>>c+d
3+4j
```

4.2.Sözcük değişkenleri:

```
>>>a = "Merhaba Python"
>>>print a
Merhaba Python
>>>b = " ile programlama"
>>>a+b
'Merhaba Python ile programlama'
```

Sözcükler için bazı fonksiyonlar vardır. Bunlara örnek olarak sözcük olan değişkeni sayıya çeviren *“int()”*, sayıyı sözcüğe çeviren *“str()”*, sözcüğün uzunluğunu veren *“len()”* verilebilir.

```
>>> a=9;b="5"
>>> print a+int(b)
14
>>> str(a)+b
'95'
>>> len(str(a)+b)
2
```

4.3.Liste değişkenleri:

Listeleri verileri tutan diziler şeklinde düşünebiliriz. Oluştururken köşeli parantez kullanılır.

```
>>> liste = ['java','c++','python']
>>> liste
['java','c++','python']
>>> liste[1]
'c++'
>>> liste = liste + ['ruby']
>>> liste
['java','c++','python','ruby']
```

Ayrıca listeler ters indistende okunabilir.

```
>>> liste[-1]
'ruby'
```

Liste elemanları aralık vererek de kullanılabilir. Burada ilk eleman verilmezse 0 kabul edilir.

```
>>> liste[1:3]
['c++','python']
>>> liste[:2]
['java','c++']
>>> liste[2:]
['python','ruby']
```

Listeler için de bazı fonksiyonlar vardır. Bunlara örnek olarak listenin sonuna eleman eklemek için *"liste.append(eleman)"*, bir elemanın listede kaç tane bulunduğunu döndürmek için

"liste.count(eleman)", iki listeyi toplamak için *"liste.extend(liste)"*, herhangi bir elemanın indisini döndürmek için *"liste.index(eleman)"*, belli bir yere eleman eklemek için *"liste.insert(indis,eleman)"*, listenin son elemanını çekmek için *"liste.pop()"*, herhangi bir elemanı listeden çıkarmak için *"liste.remove(eleman)"*, listeyi ters çevirmek için *"liste.reverse()"*, listeyi sıralamak için *"liste.sort()"*, listenin uzunluğunu döndürmek için *"liste.len()"* fonksiyonları verilebilir. Burada *"liste"* oluşturulan listenin ismi, *"eleman"* ise listedeki elemanı temsil eder. Örneğin *"liste"* olarak yularıdaki örneklerde tanımladığımız liste değişkeni, eleman olarak da *'python'* verilebilir. Örnek bir fonksiyon çağrısı:

```
>>> liste.count('java')
1
```

olabilir.

4.4.Tüp(tuple) Değişkenleri:

Tüpleri içeriği değiştirilemeyen listeler olarak düşünebilirsiniz. Parantez () kullanarak tanımlanırlar.

```
>>> diller = ('c','python')
>>> diller[1]
'python'
```

4.5.Sözlük(dictionary) Değişkenleri:

Python programcıları için sözlükler çok avantaj sağlar ve programcılar tarafından sıklıkla kullanılır. Bu tipte anahtar-sözcük ikilisi vardır ve şu şekilde tanımlanırlar:

```
sozluk={anahtar1:sözcük1,anahtar2:sözcük2,...}
```

```
>>> sozluk = {'1':'fen','2':'türkçe'}
>>> sozluk
{'1': 'fen', '2': 't\xc3\xbcrk\xc3\xa7e'}
>>> sozluk['1']
'fen'
```

Burada *'t\xc3\xbcrk\xc3\xa7e'* sizi korkutmasın python türkçe karakterleri öyle algılıyor. Yoksa türkçe karakter desteği var.

Sözlükler için de bazı fonksiyonlar vardır. Bunlara örnek olarak anahtarları döndüren *"sozluk.keys()"*, herhangi bir anahtarın sözlükte olup olmadığını tespit eden varsa 1 yoksa 0 döndüren *"sozluk.has_key(sozcuk)"*, sözcükleri döndüren *"sozluk.values()"*, bütün sozlüğü döndüren *"sozluk.items()"*, sözlükteki bir anahtara karşılık gelen sözcüğü getiren *"sozluk.get(anahtar)"*, sözlükteki bir anahtara karşılık gelen sözcüğü silen *"sozluk.del(anahtar)"*, tüm sözcükleri silen *"sozluk.clear()"*, bir sözlüğe başka bir sözlüğü ekleyen *"sozluk1.update(sozluk2)"* fonksiyonları verilebilir.

4.6.Dosyalar(files):

Dosyaları açmak için open() fonksiyonu kullanılır:

```
dosyadeğişkeni=open(dizin,mod)
```

Dosya okuma(r), yazma(w), ekleme(a), okuma ve yazma(rw) olmak üzere 4 modda açılabilir. Bazı dosya işlemleri şunlardır:

* **write(x)** : x sözcüğü dosyaya yazılır.
* **readline()** : Dosyadan bir satır okunur.
* **readlines()** : Dosyanın tamamı bir listeye okunur.
* **read(x)** : Dosyadan x byte okuma yapılır.
* **read()** : Dosyanın tamamı bir sözcüğe okunur.
* **close()** : Dosya kapatılır.

Şu şekilde bir işlem yapılabilir:

```
>>> dosyam = open('./dosya.txt', 'w')
>>> dosyam.write("ilk satır\n")
>>> dosyam.write("ikinci satır\n")
>>> dosyam.close()
```

Bu işlemden sonra python ın çalıştırıldığı dizinde vi editörüyle dosyaya bakıldığında işlemlerimizi test etmiş oluruz.

```
$vi ./dosya.txt
ilk satır
ikinci satır
```

5.KONTROL DEYİMLERİ

5.1.KARŞILAŞTIRMA OPERATÖRLERİ

Python >, >=, <, <=, ==, <>(eşit değil) gibi karşılaştırma operatörlerini gerçekler. Ayrıca lojik işlemler olarak *"and(lojik ve)"*, *"or(lojik veya)"* ve *"not(lojik değil)"* deyimlerine sahiptir.

5.2.İF DEYİMLERİ

```
if karşılaştırma:
    doğruysa çalıştırılacak blok
elif karşılaştırma:
    yürütülecek blok
else:
    yürütülecek blok
```

Yukarıdaki kullanım şekline sahip *"if"* deyimi istenen koşula göre istenen kod parçasının çalıştırılmasını sağlar.
not: Python da girintileme zorunludur. Yani aynı blokta olan kodlar aynı hizada yer almalıdır. if, for vb. bloklara girildiğinde programcı girintileme yapmak zorundadır. Eğer girintileme yapılmazsa program hata verir.

Örnek:

```
>>>if 5 > 3:
...   print "5 3 ten büyüktür."
...elif 5 < 3:
```

```
...   print "5 3 ten küçüktür."
...else:
...   print "5 3 e eşittir."
5 3 ten büyüktür
>>>
```

5.3."for" ve "while" DEYİMİ

Bu deyimlerin kullanım şekli şöyledir;

```
for degisken in liste:
    yürütülecek blok

while karşılaştırma:
    yürütülecek blok
```

"for" ve *"while"* deyimleri C ve C++ taki *"for"* ve *"while"* deyimlerinden biraz farklıdır. Hatta daha yeteneklidir. Çünkü listeler, tüpler, sözlükler üzerinde döngü yapabilme yeteneği vardır. Örnekle daha iyi görelim:

```
>>> a = ['kedi', 'kurt', 'aslan']
>>> for x in a:
...     print x, len(x)
...
kedi 4
kurt 4
aslan 5
```

Burada *"in"* kullanılır. Yani *"in"* den sonra gelen nesnenin içindeki elemanlar döngüsel bir biçimde x e atanır ve işlemler yapılır.

5.3.1."range()" Fonksiyonu

"range()" fonksiyonu belli bir aralıkta tamsayılardan oluşan bir liste oluşturmak için kullanılır. Genel yapısı şöyledir;

```
range(başlangıç, bitiş, artiş miktarı)
```

Burada başlangıç verilmezse 0 dan başlar. Artış miktarı verilmezse varsayılan değer 1 dir.Örnekle görelim:

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

5.4."break" VE "continue" DEYİMLERİ

"break" C de olduğu gibi bulunduğu for veya while deyiminden çıkmayı sağlar. *"continue"* ise bağlı bulunduğu döngünün bir sonraki aşamasına geçilmesine neden olur. Örnekle görelim:

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'eşittir', x, '*', n/x
...             break
...     else:...
...         print n, 'asal sayıdır'
...
2 asal sayıdır
3 asal sayıdır
4 eşittir 2 * 2
5 asal sayıdır
6 eşittir 2 * 3
7 asal sayıdır
8 eşittir 2 * 4
9 eşittir 3 * 3
```

5.5."pass" DEYİMİ

"pass" deyimi bir şey yapmaz. Daha önceden python' da girintileme zorunludur demiştik. Bu girintilemeyi sağlamak için eğer deyime ait blokta bir işlem yapmayacaksak *"pass"* deyimini kullanırız.

6.FONKSİYONLAR

6.1.FONKSİYON TANIMLAMA

Python da fonksiyonların genel tanımlama şekli şöyledir:

```
def fonksiyon_adı(parametreler):
    yürütülecek blok
```

Burada fonksiyon tanımlaması yapılır. Yani bu bir yürütme eylemi değildir. Fonksiyonu yürütmek için o fonksiyona çağrı yapmak gerekir. Örneklerle görelim:

```
>>> def fib(n):
...     """n ye kadar olan Fibonacci serisini yaz"""
...     a, b = 0, 1
...     while b < n:
...         print b,
...         a, b = b, a+b #aynı anda ikili atama yapılabilir
...
>>> fib(2000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Burada bu fonkison C de bildiğimiz *"void"* döndüren tipte bir fonksiyondur. Ayrıca bir değer döndüren fonksiyonlar da yazılabilir:

```
>>> def fib2(n):
...     result = []
...     a, b = 0, 1
...     while b < n:
...         result.append(b)
...         a, b = b, a+b
...     return result
...
```

```
>>> f100 = fib2(100)    # fonksiyon çağrısı
>>> f100                # fonksiyonun çıktısı
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Fonksiyon parametrelerine varsayılan değerler yüklenebilir. Bu şekilde ilgili parametre verilmediğinde varsayılan değer kabul edilir. Örnekle görelim:

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return True
        if ok in ('n', 'no', 'nop', 'nope'): return False
        retries = retries - 1
        if retries < 0: raise IOError, 'refusenik user'
        print complaint
```

Burada kullanıcıdan *"yes"* veya *"no"* girmesi isteniyor. *"raw_input"* kullanıcıdan veri alınmasını sağlar. *"raise IOError"* ise hata verilmesini sağlar. Burada varsayılan değerler *"retries"* için 4, *"complaint"* için *"'Yes or no,please!'"* dir.

Fonksiyonlar için özel fonksiyonlar da vardır. Bunlardan biri *"lambda()"* dır. Kısa olan programları def ile tanımlamak yerine anında tanımlama yapma olanağı sağlar. Örnekle görelim:

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

Burada lambda şu şablona sahiptir:

```
lambda parametre_listesi:ifade
```

6.2.FONKSİYONEL PROGRAMLAMA ARAÇLARI

Python' da fonksiyonları daha etkili kullanmak üzere 3 adet fonksiyon vardır: *filter(), map(), reduce()*.
"filter(fonksiyon, aşama)" yapısı fonksiyona aşama içindeki verileri parametre olarak atar ve aşama verisi doğruysa fonksiyonu çağırır. Daha iyi anlamak için örnekle açıklayalım:

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

Burada fonksiyon şöyle tanımlanmıştır; Eğer x 2 ye ve 3 e bölünemiyorsa x i döndür. Bu tanımlamaya göre fonksiyona parametre olarak gelen [2,25] aralığındaki sayılar *"filter()"* fonksiyonuyla f(x) fonksiyonuna parametre olarak gönderildiğinde 2 ve 3 e bölünemeyen sayılar döndürülür. Burada fonksiyon 23 kez çağrılmıştır.

"map(fonksiyon, aşama)" fonksiyonu aşama verilerinin hepsi için fonksiyonu çağırır. Sonucu bir liste şeklinde döndürür. Örnek:

```
>>> seq = range(8)
>>> def add(x, y): return x+y
…
>>> map(add, seq, seq)
[0, 2, 4, 6, 8, 10, 12, 14]
```

"reduce(fonksiyon,aşama)" fonksiyonu peşpeşe çağrılan fonksiyonların birinden dönen değeri diğerine parametre olarak verip aşamalı işlemleri yapmayı sağlar. Örnek:

```
>>> def add(x,y): return x+y
...
>>> reduce(add, range(1, 11))
55
```

Yukarıdaki işlem 1 den 10 a kadar sayıları toplamayı sağlar.

7.MODÜLLER

Program yazarken etkileşimli python kabuğunu kullanıyorsanız kabuğu kapattığınızda yazdıklarınız gider. Bunun için programlar dosyalara yazılır. Bu dosyalara fonksiyonlar tanımlanabilir. Ayrıca her fonksiyonun aynı dosyaya yazılması gerekmez. Farklı dosyalara farklı fonksiyonlar yazılabilir. Bu da modüler programcılığı oluşturur. Yani aynı işleve sahip fonksiyonları aynı dosyalara yazarak modüller oluşturulur. Burada modulün ismi dosyanın *".py"* uzantısı hariç geriye kalan isimdir. Örneğin *"fibo.py"* isminde şöyle bir python dosyası oluşturalım:

```
#vi fibo.py
def fib(n):
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Daha sonra python kabuğundan bu dosyayı modül olarak çağıralım:

```
>>> import fibo
```

Şimdi bu dosyadaki fonksiyonları kullanabiliriz;

```
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Birden fazla dosyadan oluşan bir paket programı üretiyorsak her bir dosyada kullanacağımız modülleri ekleyerek istediklerimizi gerçekleştirebiliriz. Bu C/C++ daki *"#include<>"* da benzer.

7.1."dir()" Fonksiyonu

Bu fonksiyon parametre olarak verilen modülün hangi isimleri tanımladığını sıralı liste şeklinde döndürür.

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__',
 '__stdin__', '__stdout__', '_getframe', 'api_version', 'argv',
 'builtin_module_names', 'byteorder', 'callstats', 'copyright',
 'displayhook', 'exc_clear', 'exc_info', 'exc_type', 'excepthook',
 'exec_prefix', 'executable', 'exit', 'getdefaultencoding', 'getdlopenflags',
 'getrecursionlimit', 'getrefcount', 'hexversion', 'maxint', 'maxunicode',
 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache',
 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setdlopenflags',
 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',
 'version', 'version_info', 'warnoptions']
```

8.PAKETLER

Modüller hiyerarşik bir şekilde import edilmek istendiğinde paket yapısını kullanırız. Paketler alt modüllere ayrılmış yapıya sahiptir. Örnek bir paket yapısı şu şekildedir:

```
Sound/                          #Üst düzey paket
      __init__.py
      Formats/                  #Alt paket
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      Effects/                  #Alt paket
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      Filters/                  #Alt paket
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Burada şu şekilde paketlerdeki modüller çağrılabilir:

```
>>>import Sound.Effects.echo
```

Burada Sound paketinin alt paketi olan Effects paketinden echo modülü eklenmiştir. Bu eklemeyi şu şekildede yapabiliriz:

```
>>>from Sound.Effects import echo
```

9.HATALAR ve İSTİSNALAR

9.1.SENTAKS HATALARI

Sentaks hataları daha çok programın yazımından kaynaklanan hatalardır. C/C++ a alışmış programcılar genelde Python a alışana kadar sentaks hatası yaparlar. Örneğin “while” ‘dan sonra “:” koymayınca şöyle bir hata alırız:

```
>>> while True print 'Hello world'
  File "<stdin>", line 1, in ?
    while True print 'Hello world'
                   ^
SyntaxError: invalid syntax
```

9.2.İSTİSNALAR

İstisnalar(exceptions) program çalışırken oluşan hatalardır. Bu hatalara örnek olarak sıfıra bölme hatası(division by zero) verilebilir;

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
```

9.3.HATALARLA ÇALIŞMA

Program yazarken kullanıcıdan veri aldığımızda kullanıcının istediği girişleri yapabileceğini düşünürüz. Kullanıcı yanlış veri girdiğinde hata üretmemiz gerekir. Bunu Python da şöyle yaparız;

```
>>>try:
      yürütülmek istenen blok
   except (hata_tipi):
      hata oluştuğunda yapılacak blok
```

Burada hata_tipi opsiyoneldir. Örnek olarak aşağıdaki kodu verebiliriz;

```
#vi except.py
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError, (errno, strerror):
    print "I/O error(%s): %s" % (errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
```

```
    raise
```

Yukarıdaki kodda ilk except ile “I/O Error”(Giriş – Çıkış hatası) için tanımlama yapılmıştır. Daha sonra “ValueError” (değer hatası) için tanımlama yapılmıştır. En son olarak ise bunların dışında olabilecek hatalar için tanımlama yapılmıştır.

9.4.HATA YÜKSELTME

Hata yükseltme, kullanıcının isteğine göre oluşturulan hataları yükseltmesini sağlar. Örnek:

```
>>> try:
...     raise NameError, 'Hata var!'
... except NameError:
...     print 'Hata geldi!'
...     raise
...
Hata geldi!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: Hata var!
```

Burada “NameError” olduğunda yürütülecek bir hata fonksiyonu oluşturulmuştur. “raise NameError, ‘Hata var!’” satırı ile hata fonksiyonu çağırılmıştır. Bunun sonucunda ise hata çıktısı alınmıştır.

10.SINIFLAR

Python da nesneleri oluşturmak için sınıflar kullanılır. Sınıflar değişkenlerden ve fonksiyonlardan oluşur. Genel sınıf tanımlaması şöyle yapılır:

```
class sınıf_ismi:
    sınıfa ait fonksiyonlar
```

Örnek:

```
>>> class sinif:
        def __init__(self):
            self.x=10
          self.y=20
```

Burada *"sinif"* isminde bir sınıf oluşturduk. *"__init__()"* fonksiyonu python da sınıfların yapılandırıcısı(constructor) görevindedir. Dolayısıyla sınıfı oluşturduğunuzda “*__init__()*” fonksiyonu yürütülür. Örneğe devam ederek görelim:

```
>>> a = sinif()
>>> a.x
10
>>> a.y
20
```

Görüldüğü gibi “*a*” değişkeniyle *"sinif"* ismiyle tanımlı bir sınıf oluşturuldu. Bu sınıfın yapılandırıcısı yürütüldüğünden *a.x* ve *a.y* çıktılarında daha önceden tanımladığımız değerler oluştu.

Sınıfları ilk çağırdığımızda belli parametreler ile de çağırabiliriz. Bu parametreler *"__init__()"* fonksiyonunda belirtilir. Örnekle görelim;

```
>>> class Complex:
...     def __init__(self, gercek, imajiner):
...         self.r = gercek
...         self.i = imajiner
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

Sınıflarda ayrıca miras alma(inheritance) da vardır. Miras alındığında tanımlanan sınıf miras alınan sınıfın özelliklerini de içerir. Miras alınan sınıfı üst sınıf olarak düşünürsek şu şekilde tanımlanır:

class sınıf_ismi(üst_sınıf):
sınıf içi tanımlamalar

Örnek:

```
>>> class X:
...  def __init__(self):
...     self.x=25
...  def goster(self):
...   print self.x

>>> class Y(X):
...  def __init__(self):
...   X.__init__(self)
...   self.y = 30
...
>>> a = Y()
>>> a.x
25
>>> a.y
30
>>> a.goster()
25
```

Burada görüldüğü gibi üst sınıfın yapılandırıcısının onu miras alan sınıf tarafından çağırılması gerekir. Python çoklu miras almayı(multiple inheritance) da gerçekler. O da şu şekilde gerçekleştirilir:

```
class sınıf_ismi(üst_sınıf1,üst_sınıf2,...):
      sınıf içi tanımlamalar
```

Ayrıca operatörlere yeniden yükleme(operator overloading) de yapabilirsiniz. Örneğin *"+"* operatörünün yeniden yüklemenmesi şu şekilde olur:

```
>>> class karmasik:
...  def __init__(self,g=0,i=0):
...      self.gercek=g
```

```
...       self.imaj=i
...   def __add__(self,sayi):
...       return karmasik(self.gercek+sayi.gercek,self.imaj+sayi.imaj)
...
>>> a = karmasik(3,4)
>>> b = karmasik(5,7)
>>> c = a+b
>>> print c.gercek, c.imaj
8 11
```

Kodda da görüldüğü gibi *"+"* operatörünü yeniden yüklemek için *"__add__()"* fonksiyonunu yazmak gerekiyor. Diğer operatörlerde de durum benzerdir. *"-"* için *"__coerce__()"*, *"*"* için *"__mult__()"*, *"/"* için *"__div__()"* fonksiyonlarını yazmak gerekir.

11.PYTHON'DA MYSQL VE QT İLE GUI UYGULAMASI

Şimdi python da nasıl program yazılacağını öğrendiğimize göre konunun daha iyi pekişmesi için bir örnek yapalım. Öncelikle programın kullanacağı veritabanını oluşturalım. Bu uygulama UNIX te yapılmıştır. Dolayısıyla mysql yüklü olmalıdır. Mysql işlemleri şu şekildedir;

```
cihangir$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 5.0.51a-3ubuntu5 (Ubuntu)
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> create database islem;
Query OK, 1 row affected (0.12 sec)
mysql> use islem;
Database changed
mysql> create table bilgi (ad varchar(20), soyad varchar(20),tel varchar(20),adres
varchar(40));

Query OK, 0 rows affected (0.17 sec)

mysql> insert into bilgi values ("Cihangir","Besiktas","05555555555", "EnderUNIX
Akademi");

Query OK, 1 row affected (0.39 sec)
mysql> select * from bilgi;

+----------+------------+-------------+-------------------+
| ad       | soyad      | tel         | adres             |
+----------+------------+-------------+-------------------+
| Cihangir | Besiktas | 05555555555 | EnderUNIX Akademi |
+----------+------------+-------------+-------------------+
1 row in set (0.37 sec)
```

Bu şekilde *"islem"* isimli veritabanı ve ad, soyad, tel, adres bilgisini tutan *"bilgi"* isimli veritabanı oluşturulmuştur. Şimdi bu veritabanından sorgu oluşturan gui uygulamasını yazalım. Uygulamanın kodu şu şekildedir;

```
cihangir$ vi mysql.py
# -*- coding: utf-8 -*-
```

```
import sys

from PyQt4.QtCore import *
from PyQt4.QtGui import *
import MySQLdb

class SorguPenceresi(QDialog):
    def __init__(self):
        QDialog.__init__(self)

        self.ad = QLineEdit(self)
        self.soyad = QLineEdit(self)

        self.ekle = QPushButton(u"Sorgula")
        self.ekle.setEnabled(False)
        self.iptal  = QPushButton(u"İptal")
        self.iptal.setFocusPolicy(Qt.NoFocus)

        butonlar = QHBoxLayout()
        butonlar.addStretch(1)
        butonlar.addWidget(self.ekle)
        butonlar.addWidget(self.iptal)

        gb = QGridLayout()
        gb.addWidget(QLabel(u"Ad:"), 0, 0)
        gb.addWidget(self.ad, 0, 1)
        gb.addWidget(QLabel(u"Soyad:"), 1, 0)
        gb.addWidget(self.soyad, 1, 1)
        gb.addLayout(butonlar, 2, 0, 1, 2)

        self.resize(400, 200)
        self.setLayout(gb)

        self.setWindowTitle(u"Sorgula")

        self.setTabOrder(self.ad, self.soyad)
        self.setTabOrder(self.soyad, self.ad)

        self.connect(self.ad, SIGNAL("textChanged(const QString &)"),
self.slotDenetle)
         self.connect(self.soyad, SIGNAL("textChanged(const QString &)"),
self.slotDenetle)

        self.connect(self.ekle, SIGNAL("clicked()"), self.accept)
        self.connect(self.iptal, SIGNAL("clicked()"), self.reject)
        self.connect(self, SIGNAL("accepted()"), self.sorgula)

    def sorgula(self):

        ad = (self.ad.text())
        soyad = (self.soyad.text())

        db=MySQLdb.Connection(host="localhost",user="root",passwd="LONdon23",db="is
        lem")
        cursor=db.cursor(MySQLdb.cursors.DictCursor)
         cursor.execute("select * from bilgi where ad = \"%s\" and soyad = \"%s\""
%(ad,soyad))
```

```
        R = cursor.fetchall()
        cursor.close()

        QMessageBox.information(self,u"Sorgu Sonucu",u"<center>Ad: %s<br>Soyad:
%s<br>Adres: %s<br> Tel: %s" %(R[0]['ad'], R[0]['soyad'], R[0]['adres'],
R[0]['tel']))

    def slotDenetle(self, text):
        disable = self.ad.text().isEmpty()
        disable |= self.soyad.text().isEmpty()
        self.ekle.setDisabled(disable)



def main():
    app = QApplication(sys.argv)
    pen = SorguPenceresi()
    pen.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Bu kodu yazdıktan sonra uygulamayı çalıştırmak için komut satırından şu komut çalıştırılmalıdır;

```
cihangir$ python mysql.py
```

Komut çalıştırıldıktan sonra şu şekilde bir çıktı alınır:

Sorgulanacak veri girildikten sonra “Sorgula” butonuna bastığımızda veritabanından sorgu yapıldıktan sonra şöyle bir çıktı alınır:

Şimdi kodu parça parça açıklayalım;

```
import sys

from PyQt4.QtCore import *
from PyQt4.QtGui import *
import MySQLdb
```

Öncelikle uygulama gui uygulaması olduğundan gerekli modüller eklendi. Ayrıca mysql kullanımı için MySQLdb modülü eklendi. GUI için başka modüller de mevcuttur, fakat Qt GUI için güzel bir modüldür.

```
class SorguPenceresi(QDialog):
    def __init__(self):
        Qdialog.__init__(self)
```

Daha sonra ise sınıf tanımlaması yapılır. Yapılandırıcı __init__() fonksiyonu tanımlanır. Qdialog miras olarak alındığından onun da yapılandırıcısı çağırılır.

```
        self.ad = QLineEdit(self)
        self.soyad = QLineEdit(self)
        self.ekle = QPushButton(u"Sorgula")
        self.ekle.setEnabled(False)
        self.iptal  = QPushButton(u"İptal")
        self.iptal.setFocusPolicy(Qt.NoFocus)

        butonlar = QHBoxLayout()
        butonlar.addStretch(1)
        butonlar.addWidget(self.ekle)
        butonlar.addWidget(self.iptal)

        gb = QGridLayout()
        gb.addWidget(QLabel(u"Ad:"), 0, 0)
        gb.addWidget(self.ad, 0, 1)
        gb.addWidget(QLabel(u"Soyad:"), 1, 0)
        gb.addWidget(self.soyad, 1, 1)
        gb.addLayout(butonlar, 2, 0, 1, 2)
```

```
        self.resize(400, 200)
        self.setLayout(gb)

        self.setWindowTitle(u"Sorgula")

        self.setTabOrder(self.ad, self.soyad)
        self.setTabOrder(self.soyad, self.ad)
```

Yukarıda ise GUI arabirimleri tanımlanmıştır.

```
        self.connect(self.ad, SIGNAL("textChanged(const QString &)"),
self.slotDenetle)
        self.connect(self.soyad, SIGNAL("textChanged(const QString &)"),
self.slotDenetle)

        self.connect(self.ekle, SIGNAL("clicked()"), self.accept)
        self.connect(self.iptal, SIGNAL("clicked()"), self.reject)
        self.connect(self, SIGNAL("accepted()"), self.sorgula)
```

Daha sonra bu arabirimler üzerinde işlem yapıldığında yürütülecek işlemler belirtilmiştir. Örneğin son üç satırda “Sorgula” tuşuna basıldığında “accept” in çalıştırılcağı, “accept” de çalıştırıldığında sorgu işlemlerini gerçekleştirecek “sorgula” fonksiyonunun çalıştırılacağı belirtilmiştir.

```
   def sorgula(self):

         ad = (self.ad.text())
         soyad = (self.soyad.text())

         db=MySQLdb.Connection(host="localhost",user="root",passwd="",db="islem")
         cursor=db.cursor(MySQLdb.cursors.DictCursor)
         cursor.execute("select * from bilgi where ad = \"%s\" and soyad = \"%s\""
%(ad,soyad))
         R = cursor.fetchall()
         cursor.close()

         QMessageBox.information(self,u"Sorgu Sonucu",u"<center>Ad: %s<br>Soyad:
%s<br>Adres: %s<br> Tel: %s" %(R[0]['ad'], R[0]['soyad'], R[0]['adres'],
R[0]['tel']))
```

Yukarıda ise sorgula fonksiyonu tanımlanmıştır. Bu fonksiyon MySQLdb paketindeki fonksiyonları kullanarak önce mysql bağlantısı yapmış, daha sonra bir sorgu gerçekleştirmiş, daha sonra ise bağlantıyı kapatmıştır. Fonksiyonun sonunda ise *“QmessageBox”* ile ekrana mysql sorgusundan dönen sonuçlar basılmıştır.

```
  def slotDenetle(self, text):
        disable = self.ad.text().isEmpty()
        disable |= self.soyad.text().isEmpty()
        self.ekle.setDisabled(disable)
```

Yukarıdaki “slotDenetle” fonksiyonu ise kullanıcıdan girişleri aldıktan sonra “Sorgula” butonunu etkin hale getiren kodu içermektedir.

```
def main():
    app = QApplication(sys.argv)
    pen = SorguPenceresi()
    pen.show()
    sys.exit(app.exec_())
```

Burada ise “main” fonksiyonu tanımlanmıştır. Bu fonksiyon programın çalışmasını ve ilgili GUI elemanlarının gösterilmesini sağlamaktadır.

```
if __name__ == "__main__":
    main()
```

Program çalıştırıldığında ise yukarıdaki *“if”* bloğu *“true”* olacaktır ve program çalışacaktır.

Bu program daha çok pythonda sınıf tanımlamaları, miras alma, fonksiyon tanımlamaları vb. nin nasıl yapıldığını göstermek için yazılmıştır. Yoksa bu programla çok detaylı bir GUI anlatımı amaçlanmamaktadır. GUI uygulaması geliştirmek için PyQt kullanmak isteyenler daha detaylı bilgileri internetten bulabilirler.

KAYNAKLAR

- **www.python.org/doc**
- **The PyQt4 Tutorial http://www.zetcode.com/tutorials/pyqt4/**