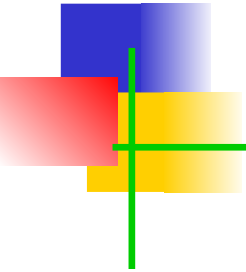


# BLM442 Büyük Veri Analizine Giriş

## Python Programlama

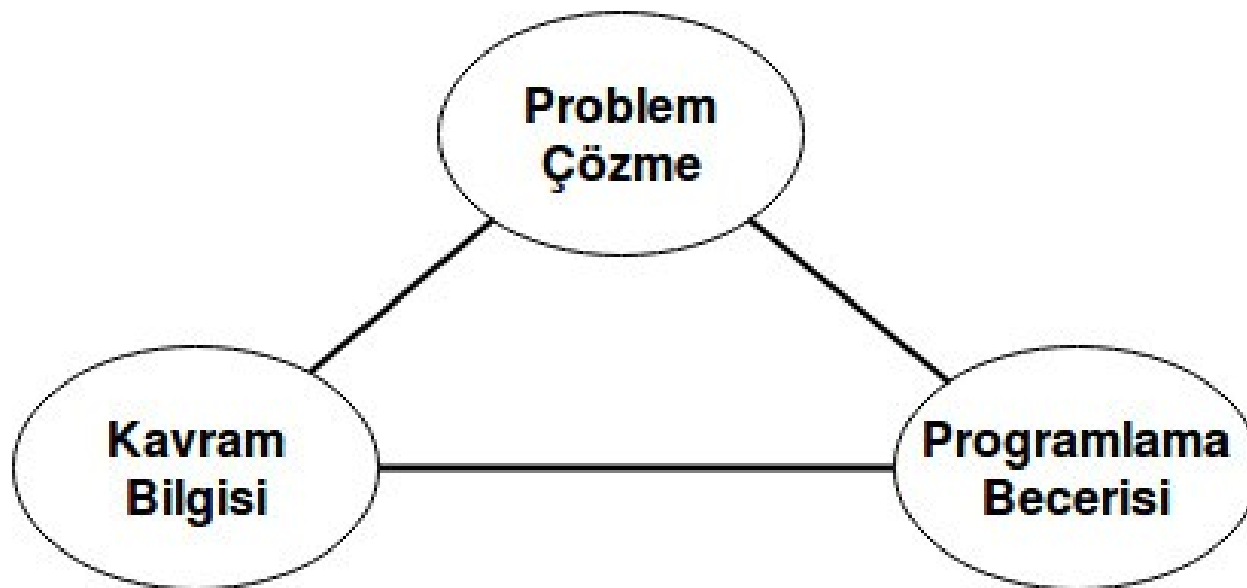


Dr. Süleyman Eken

Bilgisayar Mühendisliği  
Kocaeli Üniversitesi

# Sunum Planı

- Modüller, aritmetik, fonksiyonlar
- Strings, lists, tuples, dictionaries, sets veri yapıları
- Koşul ifadeler ve döngüler
- Sıralama, list comprehensions, generators, iterators
- Nesne yönelimli programlama
- Functional tools, enumerate, zip vs.



# Bir bilgisayar ne yapar?

- Saniyede bir milyon hesaplama yapar
- 100 gigabaytlık depolama sonuçlarını hatırlar
- Ne tür hesaplamalar?
  - dil içindeki tümleşik (built-in) olanlar
  - programcı olarak bizim tanımladıklarımız
- Bilgisayar biz ne söylersek onu anlar

# Kavram türleri

- Açıklayıcı (declarative) kavram,  
bir gerçeğin ifadeleridir  
tanımlamalardır
  - $x$  sayısının karekökü  $y$  ise  $y^2=x$  olmalı
- Emirli (imperative) kavram,  
bir şeyin nasıl yapılacağı ifadeleridir.  
bir tariftir (basit emirler dizisi, kontrol akışı, sonlandırma vs)

## Babylonian method

Get  $x$  as an input

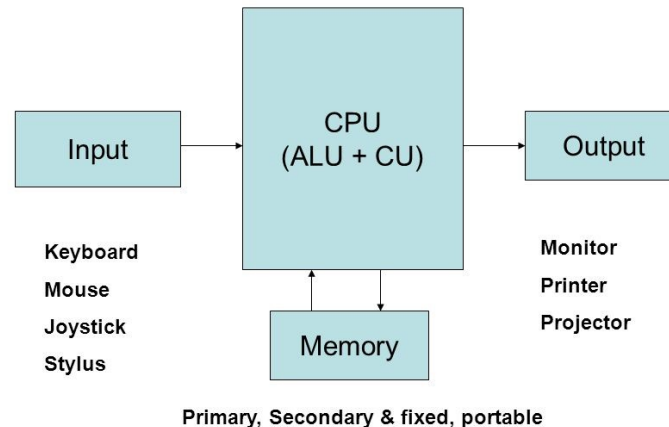
1. Begin with an arbitrary positive number  $y_0$  *(an initial guess)*
2. If  $y_n^2 \approx x$ , stop *(found the solution -  $y_n$ )*  
Else let  $y_{n+1} = (y_n + x/y_n)/2$  *(use the arithmetic mean to approximate the geometric mean)*
3. Repeat step (2)

$g$	$g^2$	$x/g$	$(g+x/g) / 2$
3	9	16/3	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

# Bilgisayar=makine

- Mekanik bir sistem içinde bir tarif/problem çözümü nasıl olur?
  - Sabit programlı makine (fixed program): hesap makinesi
  - Depolanmış programlı makine (stored program): emirler saklanır ve yorumlayıcı ile her emri sırası ile icra edilir.

A typical computer



# Bir programlama dilinin boyutları

## Düşük seviye Vs. Yüksek seviye

- Soyutlama (abstraction) seviyesine göre ayırım
- Düşük seviyeli programlama dillerinde (örneğin Assembly), hesaplamalarda kullanılan emirler (instructions) çok basit (yaklaşık makine düzeyinde)
- Yüksek seviyeli (örneğin Python, C, Java) bilgisayarın donanım mimarisinden bağımsız olarak kullanıcı dostu yazılımlar oluşturmaya yardımcı olan programlama dilleridir.

# Bir programlama dilinin boyutları

## General Vs. Targeted

- Uygulama kapsamına göre ayırım
- Genel bir programlama dilinde, dilin sağlamış olduğu temel operasyonlar (primitives) ve daha karmaşıkları (fonksiyonlar vs) çok çeşitli uygulamaları destekler.
- Hedeflenen bir programlama dili, çok özel bir dizi uygulamaya yöneliktir.
  - Örneğin MATLAB (matrix laboratory) matris ve vektör sayısal işlemleri için özel tasarlanmış programlama dili



# Bir programlama dilinin boyutları

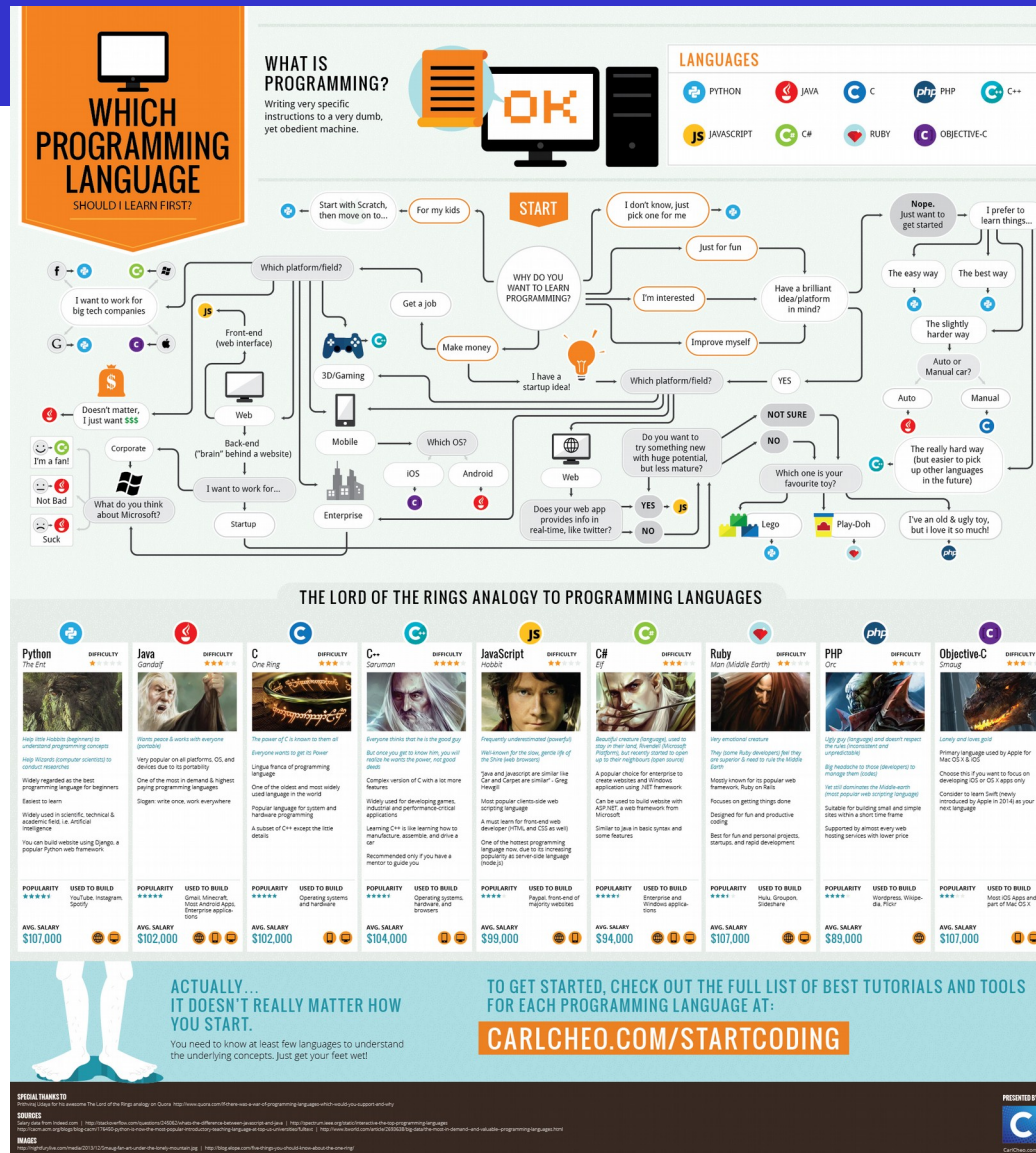
## Interpreted vs. Compiled

- Kaynak kodun nasıl çalıştırıldığına (executed) göre ayırım
- Yorumlanan dillerde (örneğin LISP) kaynak kod direk olarak çalışma zamanında yorumlayıcı tarafından çalıştırılabilir koda dönüştürülür.
  - Yorumlayıcı, kodda hatalı satıra gelene kadar size bir şey söylemez gerekli yorumlama işlemlerini yapar ve o satıra geldiğinde hata çıktısını gösterir.
- Derlenen dillerde (örneğin C), kaynak kodun çalıştırılabilir hale getirilmesinden önce derleyici tarafından bir nesne koduna çevrilmesi gerekir.

# Programlama dilleri paradigmaları

- Fonksiyonel
  - Lisp, Scheme, Haskell
- Emirli programlama (Imperative)
  - FORTRAN, BASIC, Pascal, C
- Mantıksal/tanımlamalı (logical/declarative)
  - Prolog
- Nesne yönelimli
  - C++, Java, C#, Python

# Hangi programlama dilini öğrenmeliyim?







































# 2018 Top programlama dilleri

Language Types (click to hide)





























Language Rank    Types    Spectrum Ranking

1. Python	   	100.0
2. Java	  	97.5
3. C#	  	89.4
4. PHP		84.9
5. JavaScript	 	82.6
6. Go	 	76.4
7. Scala	 	72.1
8. Ruby	 	71.4
9. HTML		71.2
10. Perl	 	57.4
11. Processing	 	53.1
12. Lua	 	49.8
13. Rust	 	41.8
14. D	 	40.6
15. Clojure	 	25.6
16. Ocaml	 	14.4
17. Actionscript	 	0.6

Language Types (click to hide)

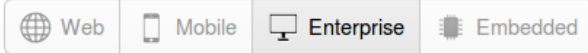


Language Rank    Types    Spectrum Ranking

1. C++	  	99.7
2. Java	  	97.5
3. C	  	96.7
4. C#	  	89.4
5. JavaScript	 	82.6
6. Scala	 	72.1
7. Swift	 	53.9
8. Objective-C	 	50.5
9. Delphi	 	38.7
10. Scheme	 	18.8
11. Actionscript	 	0.6

# 2018 Top programlama dilleri

## Language Types (click to hide)



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. Java		97.5
4. C		96.7
5. C#		89.4
6. R		82.9
7. Go		76.4
8. Matlab		72.8
9. Ruby		71.4
10. Shell		66.1
11. Perl		57.4
12. Swift		53.9
13. Processing		53.1
14. Objective-C		50.5
15. Lua		49.8
16. Fortran		49.5
17. SQL		49.3
18. Haskell		48.6
19. Visual Basic		45.1

## Language Types (click to hide)



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		99.7
3. C		96.7
4. Assembly		74.1
5. Arduino		69.0
6. Haskell		48.6
7. VHDL		45.4
8. Verilog		41.2
9. D		40.6
10. LabView		32.7
11. Erlang		26.9
12. TCL		21.9
13. Ada		20.9
14. Ladder Logic		11.5
15. Forth		0.0

<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>

# Python yorumlayıcısı

- Standart interaktif Python yorumlayıcı terminale “python” yazarak çağrılabilir.
- `$ python`  
ipcvlab@ipcvlab ~ `$ python`  
Python 2.7.14 |Anaconda custom (64-bit)| (default, Dec 7 2017, 17:05:42)  
[GCC 7.2.0] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
`>>> a=5`  
`>>> print a`  
5
- Çıkış için `exit()` veya `Ctrl-D`  
ipcvlab@ipcvlab ~ `$ python hello_world.py`  
Hello Python
- `ipcvlab@ipcvlab ~ $ jupyter notebook`

# Python yorumlayıcısı - Ipython Shell

- ipcvlab@ipcvlab ~ \$ ipython  
Python 2.7.14 |Anaconda custom (64-bit)| (default, Dec 7 2017, 17:05:42)  
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.

? -> Introduction and overview of IPython's features.

%quickref -> Quick reference.

help -> Python's own help system.

object? -> Details about 'object', use 'object??' for extra details.

```
In [1]: a=5
```

```
In [2]: a
```

```
Out[2]: 5
```

- Tab ile tamamlama (completion)

```
In [3]: an_apple=27
```

```
In [4]: an_example=34
```

```
In [5]: an<Tab>
```

```
an_apple    and
```

```
an_example  any
```

```
In [6]: b=[1,2,3]
```

```
In [7]: b.<Tab>
```

```
b.append b.index b.remove
```

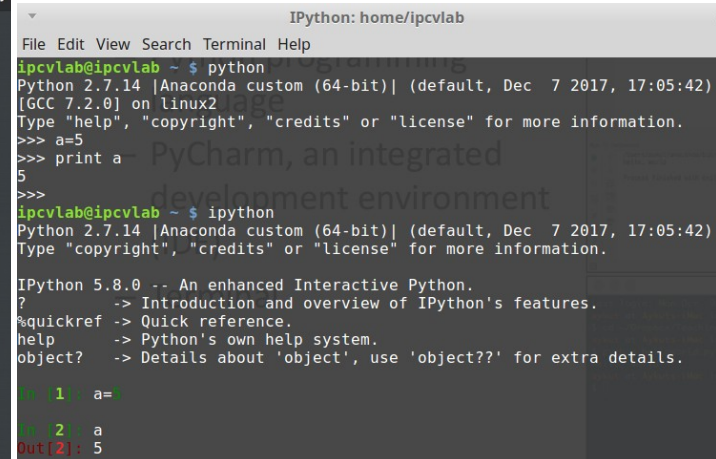
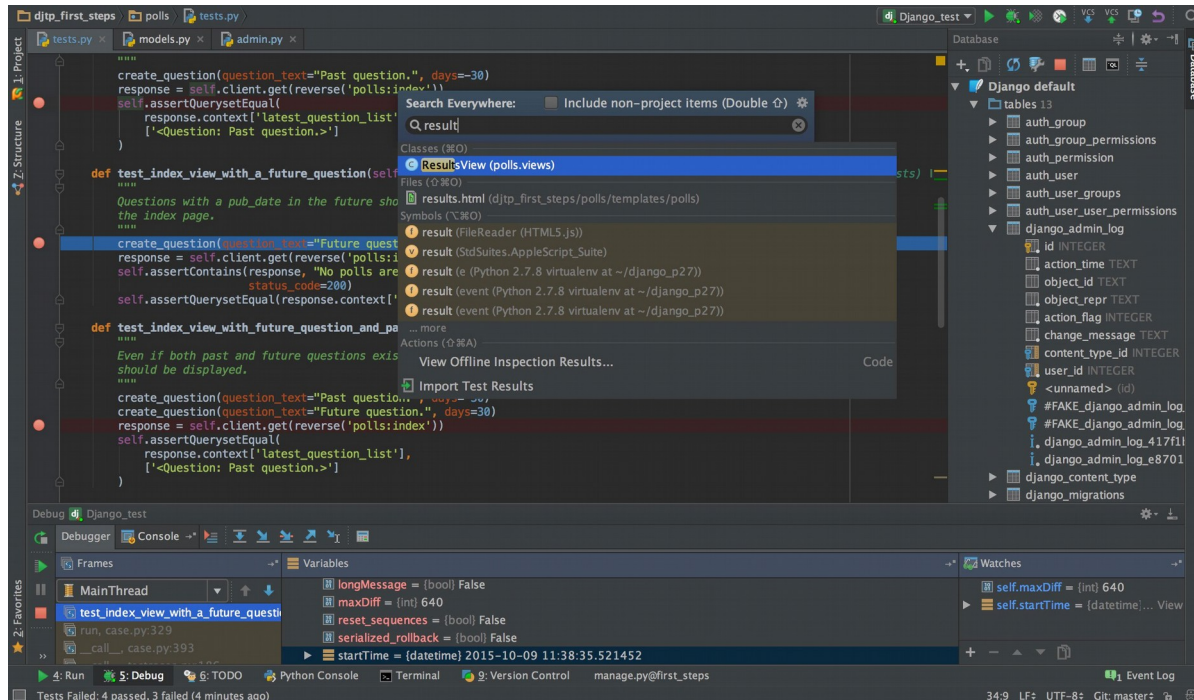
```
b.count b.insert b.reverse
```

```
b.extend b.pop b.sort
```



# Python programlama

- Entegre gelistirme ortamı (IDE): PyCharm, Spyder
- Terminal



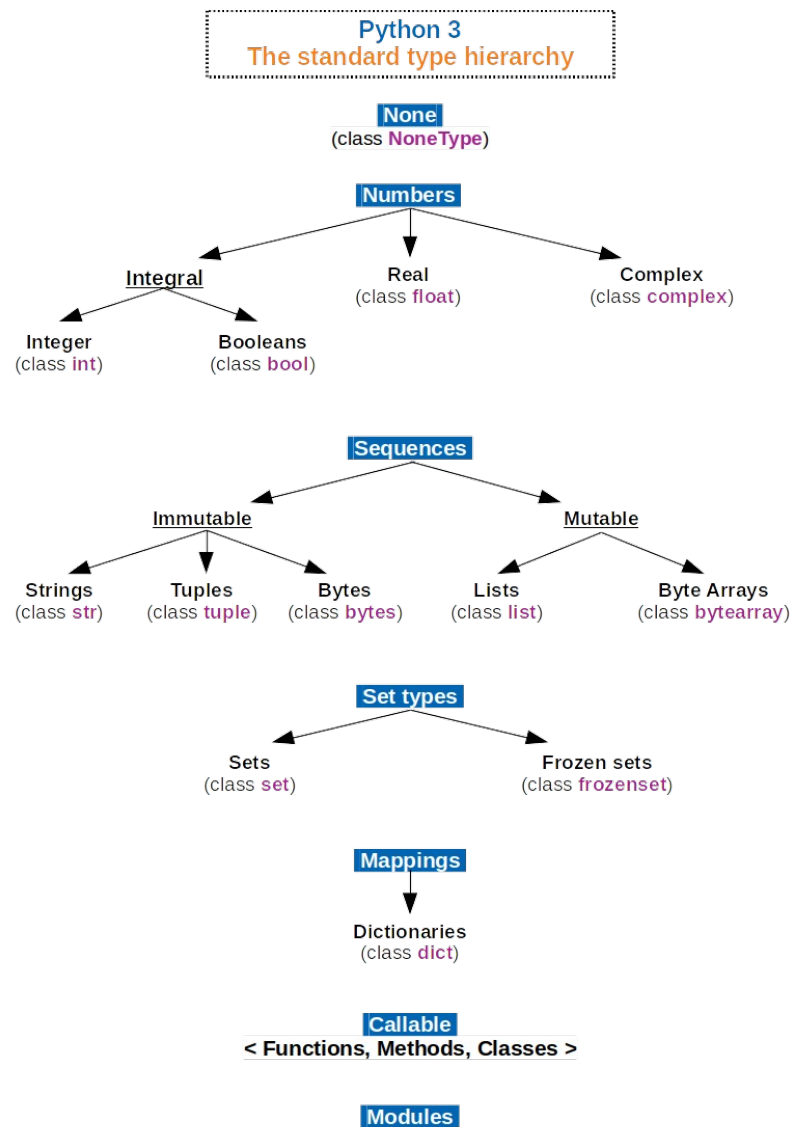


# Giriş-çıkış

- Giriş tipleri: komut satırı argümanları, standart giriş, dosya giriş
- Çıkış tipleri: standart çıkış, dosya çıkış, grafiksel çıkış, ses çıkış
- `$ python my_program.py arg_1 arg_2 ... arg_n`
- array şeklinde program içinde erişmek mümkün
  - `sys.argv[1], sys.argv[2], . . . , sys.argv[n]`
- Program ismi (`my_program.py`) `sys.argv[0]` tutulur.



# Python 3 standart tip hiyerarşisi



# Booleans: yeni tip

- True, False.
- operations: not, and, or.

x	y	not x	not y	x and y	x or y
True	True	False	False	True	True
True	False	False	True	False	True
False	True	True	False	False	True
False	False	True	True	False	False

- Boolean değerlerini adaptif olarak atamak istiyorsak ne olur?
  - relational operatorler  $<$ ,  $>$ ,  $\leq$ ,  $\geq$ ,  $\neq$ ,  $==$  kullanılır
  - $5+7 < 4*3$  or  $1-2 > 2-4$  and  $15 == 4$  geçerli ifadeler
  - aritmetik  $>$  relational  $>$  boolean öncelik sırası

# If ifadeleri

- genel form

```
if condition:  
    block
```

- Örnek:

```
if grade >=50:  
    print "pass"
```

- Condition, Bool bir ifadedir.

- Diğer form

```
if condition:  
    block  
else:  
    block
```

- Daha genel form

```
if condition1:  
    block  
elif condition2:  
    block  
elif condition3:  
    block  
else:  
    block
```

# Fonksiyonlar

- Güncelleme ve test işlemlerini çok daha kolay hale getiren kod parçalarını tekrar kullanmamıza izin verir.
- Bilgileri daha iyi ayrıştırmamızı sağlar.
- Geri donus:
  - return ifadesi ile fonksiyon sonlanır ve sonrasındaki ifadenin değeri geri döndürülür.
  - return ifadesi yoksa, fonksiyon None döndürür.
  - print, virgülle ayrılmış bir veya daha fazla ifadeyi alır ve bunları ekrana yazdırır.
  - print, return ifadesinden farklı; ancak shell'de aynı görünüyor.

# Çoklu fonksiyon çağırma

- Bazen başka fonksiyonları çağıran fonksiyonlara sahip olmak istiyoruz.
  - $f(g(4))$
  - Bu durumda, 'inside out' kuralını kullanırız, yani önce  $g$  koşar, sonra  $f$  sonucu araştırılır.

# Fonksiyon dizaynı

- Fonksiyonun neyi bilmesi gerektiği, parametre olarak verilmeli
- Geri dönüş var mı
- Dokümentasyonu
  - Fonsiyonlar veya modüller hakkında bilgi almak için built-in fonksiyon `help()` kullanabiliyoruz. `help(zip)` gibi
  - docstrings ile tanımladığımız fonksiyona yardım bilgileri ekleyebiliriz.
  - Bir docstring `'''` ile çevrilidir ve bir modül veya fonksiyonun ilk satırı olmalıdır.

# Docstrings

- Bir modül veya fonksiyonun ilk satırı bir string ise, onu docstrings olarak adlandırırız.
- Python `help()` çağrıldığında döndürülecek string'i kaydeder.
- Tüm fonksiyonların docstrings olmalıdır.
- Taşınabilirliği ve güncellemeyi kolaylaştırır.
- Diğer kişilerin, kodunuza girmeden fonksiyonlarınızın ne yaptığını ve nasıl kullanılacağını bilmelerini sağlar.
- Bir fonksiyonun nasıl çalıştığını tanımlamaz.
- Her parametrenin amacını netleştirir ve parametreye ada göre işaret eder.



# Örnek Docstrings

- `def square(a):`  
    `"""Returns argument a is squared."""`  
    `return a**a`
- `print (square.__doc__)`
  - Returns argument a is squared.
- `help(square)`
  - Help on function square in module `__main__`:  
    `square(a)`  
        Returns argument a is squared.
- <https://www.datacamp.com/community/tutorials/docstrings-python#comments>

Formatting Type	Description
<a href="#">NumPy/SciPy docstrings</a>	Combination of reStructured and GoogleDocstrings and supported by Sphinx
<a href="#">Pydoc</a>	Standard documentation module for Python and supported by Sphinx
<a href="#">Epydoc</a>	Render Epytext as series of HTML documents and a tool for generating API documentation for Python modules based on their Docstrings
<a href="#">Google Docstrings</a>	Google's Style

# Strings

- Karakter dizileri
- Strings tek veya çift tırnak işareti ile gösterilir.
- String'ler toplanabilir (concatenation)
  - “str” + “ing” → “string”
- Çarpılabilir
  - “copy”\*3 → “copycopycopy”
- Ayrıca ilişkisel operatörleri kullanarak karşılaştırabilir.
- Alt string'lerin “in” kullanarak bir string içinde olup olmadığı kontrol edebilir.
- Birden çok satıra yayılan uzun string'ler ''' (üç tırnak) kullanılarak tanımlanabilir.
- Escape karakterleri
  - \n - a new line
  - \' - a single quote
  - \" - a double quote
  - \\ - a backslash
  - \t - a tab.
- len(string)

# String'e cevir

- String olmayan bir değişkenimiz varsa ve string'e dönüştürmek istiyorsak:
  - `str(x)` → x'i string cevirir
- `print`, farklı tipleri görüntüleyebilir.
  - `print "Person", name, "has height", height, "age", age, "weight", weight`
- Bir tip eklemek için `print` içinde özel karakterler (`%d`, `%s`, `%.2f` vs) kullanılabilir.
  - `print "My age is %d." % age`
  - `print "Person %s has weight %.2f and age %d and height %d." \`  
`% (name, weight, age, height)`

# Modüller

- Bazen başkalarının kodunu kullanmak istiyoruz.
- Veya kendi kodumuzu kullanıma hazır hale getirin.
- İlgili fonksiyonları bir dosyada toplamak gerekiyor.
- Modüller bunu yapmamıza izin veriyor.
- Modül, ilgili fonksiyonlar ve değişkenlerden oluşan bir gruptur.

# Modül kullanımı

- Bir modül kullanmak için, onu import etmek gerekir.
- Bir modülün import edilmesi python'ın modüldeki her kod satırını çalıştırmasını sağlar.
- Bir modüldeki fonksiyonu kullanmak için
  - `module_name.function_name()`
- ya da modülü bir kere çalıştırıp fonksiyon direk kullanılır
  - `function_name()`
- Başka bir şekilde
  - `from module_name import fn_name1(), fn_name2()`
  - artık `fn_name1()` şeklinde refer edilebilir kod içinde
  - `from module_name import *` tüm fonk. import için

# String metotları

- `help(str)` şeklinde hangi metotlar var bakılabilir veya
- `dir( functions, modules, strings, lists, dictionaries vs.)` `dir(str)` gibi
- Birkaçı:
  - `s.replace(old, new)` - string s içindeki old patterni new ile değiştirilir
  - `string.count(substr)` – string içinde kaç tane substr var
  - `string.lower()` - büyükce çevir
  - `string.upper()` - küçükce çevir
  - <https://docs.python.org/3/library/>

# while döngüsü

*while condition:*

*block*

- Süresiz olarak bir işlemi tekrarlamak istersiniz
- Bir koşul yerine getirilinceye kadar bir işlemi tekrarlamak istiyorsunuz.
- Bir eylemi sabit sayıda tekrarlamak istiyorsunuz.

# Degismez ve deđiřir nesneler (Immutable vs. mutable objects)

- Python'daki her řey bir nesne olduđundan, her deđiřken bir nesne rneđini (instance) tutar.
- Bir nesne ilklendirildiđinde, benzersiz bir nesne kimliđi (id) atanır. Tipi alıřma zamanında tanımlanır ve bir kez set edildikten sonra asla deđiřiklik yapılamaz (immutable), ancak durumu deđiřtirilebiliyorsa deđiřir (mutable) nesnelerdir.
- `tuple1 = (0, 1, 2, 3)`  
`tuple1[0] = 4`  
`print(tuple1)`  
TypeError: 'tuple' object does not support item assignment
- `color = ["red", "blue", "green"]`  
`color[0] = "pink"`  
`color[-1] = "orange"`  
`['pink', 'blue', 'orange']`
- **Mutable objects:** list, dict, set, byte array
- **Immutable objects:** int, float, bool, complex, string, tuple, frozen set [note: immutable version of set], bytes



# Fonksiyonlara nesneler nasıl aktarılır?

- Değiştirilen nesneler referans ile (called by reference) çağrılabilir, orijinal değişkenin kendisini değiştirebilir.

```
def updateList(list1):  
    list1 += [10]  
  
n = [5, 6]  
print(id(n))                # 140312184155336  
  
updateList(n)  
print(n)                    # [5, 6, 10]  
print(id(n))                # 140312184155336
```

- Değişmez nesneler fonksiyona arguman olarak aktarıldığında (değer ile çağırma/called by value) nesnenin kendisi değil sadece değişkenin değeri iletilir.

```
def updateNumber(n):  
    print(id(n))  
    n += 10  
  
b = 5  
print(id(b))                # 10055680  
updateNumber(b)             # 10055680  
print(b)                    # 5
```

# Listeler

- `list_name = [list_elt0, list_elt1, ..., list_eltn]`
- `i.` elemana erismek
  - `list_name[i-1]`
- Bos liste: `[]`
- `list_name[-i]` sondan elemana erisim
- `list_name[-1]` sonuncu eleman
- Listeler heterojen yapılıdır.
  - `int`, `string`, hatta baska bir liste olabilir

# Liste fonksiyonlar-metotlar

- **len(list\_name)** → liste uzunlugu
- **min(list\_name)** and **max(list\_name)**
- **sum(list\_name)** → sayı degerlerli elemanlar icin toplam doner.
- **append(value)** – adds the value to the end of the list.
- **sort()** - sorts the list so long as this is well defined.  
(need consistent notions of > and ==)
- **insert(index, value)** – inserts the element value at the index specified.
- **remove(value)** – removes the first instance of value.
- **count(value)** – counts the number of instances of value in the list.

# Listeler üzerinde döngü

- Listenin her elemanı üzerinde aynı işlem yapılacaksa

```
for item in list:  
    block
```

- **range(i)** returns an ordered list of ints ranging from 0 to i-1.
- **range(i, j)** returns an ordered list of ints ranging from i to j-1 inclusive.
- **range(i, j, k)** returns a list of ints ranging from i to j-1 with a step of at least k between ints.

```
for i in range(len(list)):  
    block
```

## Liste dilimleme (slicing)

- Bazen listenin bir kısmı üzerinde işlem gerekir.
- Dilimleme youyla yeni bir liste olur.
- `sublist=list slicing`
- `y=x[i:j]` i. indis ile j-1. indis arasındaki listeyi döner
  - `x[:]` orjinal listenin tüm elemanları
  - `x[i:]` i.indisten sona kadar (son dahil)
  - `x[:j]` baştan j-1. indise kadar

# List comprehensions

- Bir collection elemanları filtreleme yoluyla liste oluşturma
- `[expr for val in collection if condition]`
- `result = [ ]`  
for val in collection:  
    if condition:  
        result.append(expr)
- `strings=['a', 'as', 'bat', 'car', 'dove', 'python']`  
In [155]: `[x.upper() for x in strings if len(x)>2]`  
Out [155]: `['BAT', 'CAR', 'DOVE', 'PYTHON']`

# String'lere geri donelim

- `find(substring)`: give the index of the first character in a matching the substring from the left or -1 if no such character exists.
- `rfind(substring)`: same as above, but from the right.
- `find(substring,i,j)`: same as `find()`, but looks only in `string[i:j]`.
- `a="gjughjghj" idi`
  - In [40]: `a.find('j')`
  - Out[40]: 1
  - In [41]: `a.rfind('j')`
  - Out[41]: 8
  - In [42]: `a.find('j',4,7)`
  - Out[42]: 5

# İç içe listeler

- Bu, matrisler istiyorsak ya da daha yüksek boyutlu bir alanı temsil ediyorsak kullanışlıdır.
- i. listenin j. elemanının erismek için
  - `list_name[i][j]`
- Eğer tüm elementlerin üzerinde döngü yapmak istiyorsak, iç içe döngülere ihtiyacımız var:

```
for item in list_name:  
    for item2 in item:  
        block
```



# Listeler ek konular

- Stringler de olduğu gibi + ve \* kullanımı vardır
- + listeleri toplar
  - In [44]: c=[2,3,4]  
In [45]: d=[5,6,7]  
In [46]: c+d  
Out[46]: [2, 3, 4, 5, 6, 7]
- \*n n-tane liste kopyalar
  - In [50]: c\*3  
Out[50]: [2, 3, 4, 2, 3, 4, 2, 3, 4]
- **x in list** is a boolean operation that tests if x is in the list.
- **list.pop()** removes and returns the last item on the list.

# Tuples

- Listelerin degismez olması=tuples  
`tuple_name=(item0,item1,item2,...)`
- Bos tuple
  - `()`
- Tuple'ın i. Indisindeki elemana `tuple_name[i]` ile erisilir  
`tuple_name(i)` seklinde degil
- Tek elemanlı tuple virgul ile tanımlanır
  - `(12,)`
- Strings ayrı karakterlerin tuple'ı olarak kabul edilebilir (değişmez oldukları için).
  - `min()` ve `max()` strings üzerinde tanımlı iken `sum()` degildir
  - string'ler üzerinde dilimleme yapılabilir.
  - In [36]: `a="gjughjghj"`
  - In [37]: `a[2:6]`
  - Out[37]: `'ughj'`

# Liste kısıtları

- Liste üzerinde eleman aramak zaman alır.
- Ya istedikimiz/aradığımız veriler hakkında fazla bir şey bilmiyorsak?
- Veriye göre indeksleme yok
  - no `list_name[data]`, sadece `list_name[i]`

# Dictionaries

- Sozlukler (key,value) çiftinden oluşur.
- maps diye de adlandırılır.  
`{key0 : value0, key1 : value1, ..., keyn : valuen}`
- dict tipindedirler.
- key ile ilişkilendirilmiş bir değer bulmak için  
`dictionary_name[key]`
- `dictionary_name[value]` seklinde ters erişim yok
- Sözlükler sıralanmamış (unsorted).
- Sözlük “key” değerleri değişmez olmalıdır (immutable), ancak “value” herhangi bir şey olabilir.
- Sözlük tanımlandıktan sonra key-value çifti eklenebilir.  
`dictionary_name[key] = value`
- key degerleri uniq olmalı

# Dictionary metotları

- `len(dict_name)` uzunluğunu verir
- `+` and `*` tanımlı değil
- `dict.keys()` - key değerleri gelir
- `dict.values()` - value değerleri gelir
- `dict.items()` - (key, value) çiftleri gelir
- `dict.has_key(key)` – eğer sözlükte ilgili key varsa **True** döner
- `dict.get(key)` – key karşılık gelen value döner, key yoksa **None** döner
- `dict.clear()` - sözlükten tüm (key, value) çiftlerini siler
- `dict.copy()` - tüm sözlüğü kopyalar
- `dict.pop(key)` – key ile indekslenen key-value çiftini sözlükten siler ve onu geri döndürür

# Dictionary üzerinde döngü

```
for key in d:  
    print key, d[key]
```

- Çalışır fakat yavaş

```
for key in d.iterkeys():  
    print key, d[key]
```

- Daha iyi

# Dosya işlemleri

- Python'un dosyalarla uğraşmak için kullanılan bir türü vardır.
- Dosya açma, okuma, yazma, kapama
- Dosya ismi;
  - kod içinde hard-coded, `raw_input()` ile kullanıcıya sorarak
  - bazı modül (media) `choose_file()` dialog penceresinden secerek
- İsim olduktan sonra dosya acma
  - `open(filename, 'r')` – for reading (this is the default mode).
  - `open(filename, 'w')` – for writing (erases the contents of a file).
  - `open(filename, 'a')` – for appending (keeps the contents of the file).
  - fonksiyon bir file nesnesi doner

# Dosya okuma

- Tüm dosyanın icini string olarak doner, büyük dosyalar icin tavsiye edilmez

```
filename.read()
```

- Satır satır okur

```
filename.readline()
```

- 5. satırı okumak icin

```
filename.read(5)
```

- Her adımda bir satır doner

```
for line in filename:  
    print line
```



# Dosyaya yazma ve kapama

- Yazmak için

```
filename.write("This is a  
string")
```

- Çoklu yazma ile string concatenation olur
- Dosyaya yazmak için yazma veya ekleme (append) modda acmak gerek
- Append mode'da string dosya sonuna eklenir.
- Kapamak için

```
filename.close()
```

# Classes

- Python, kendi türlerimizi oluşturmamızı sağlar.
- Bunlara sınıf denir.
- Bir sınıfımız olduğunda, o sınıfın örneklerini / nesnelerini oluşturabiliriz.
- Sınıf oluşturmak için

```
class Class_name(object) :  
    block
```

- object tipi gösterir, class ise keyword'dur
- Sınıf isimleri büyük harfle başlar
- Sınıf metotları ve örnekleri küçük harfle başlar
- Sınıf tanımlamaları modüller gibi sınıfın ne olduğunu göstermesi açısından docstrings içermelidir.
- Nesne oluşturmak için

```
x = Class_name()
```

## Sınıf metotları

- Sınıf nesneleri tek başına birşey yapamaz.
- Onları daha kullanışlı hale getirmenin bir yolu, sınıf nesnelere metotlar eklemektir.
- Bunu sınıf adının altındaki kod bloğuna koyarak yapıyoruz.
- Fakat metotlarımızın her bir sınıf nesnesi üzerinde çalışmasını istiyoruz.
- Yöntemimizi tüm sınıflardan ziyade sınıf nesnemizde çalıştırmamız gerekli.
- self anahtar kelimesi ile yapıyoruz

# Sınıf metotları

```
class Patient(object):  
    def set_age(self, age):  
        self.age = age
```

- `p1` nesnesine yas ataması için
  - `p1.set_age(10)` seklinde olur `p1.set_age(p1, 10)` boyle degil
- Sınıf yapıcısı (constructors) özel `__init__` metodu ile gerceklenir.

```
class Patient(object):  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

- `x = Patient("Joey", 10)`
- Metot tanımları, fonksiyon tanımları gibi docstrings içermelidir.

# Sınıf özel metotları

- `__` ile başlaması özel olduğu anlamına gelir.
  - `__str__` is used when printing
  - `__cmp__` is used to allow boolean operations.
  - `__add__` is used to allow the + operator.
  - `__iter__` is used to allow your type to be used in for loops.

# Classes - Encapsulation

- Sınıfların en büyük yararlarından biri, uygulama detaylarını kullanıcıdan gizlemeleridir.
- Biz buna kapsülleme diyoruz.
- İyi tasarlanmış bir sınıf, kullanıcının ihtiyaç duyduğu tüm bilgileri öğrenmesini sağlayan yöntemlere sahiptir.

# Classes - Inheritance

- Var olan kodun değiştirilmesine izin veren, var olan kodun çalıştırılma özelliğini değiştirmeyen bir yöntem istiyoruz.
- Bunu yapabilmemizin bir yolu, eski sınıfı kopyalayan ve bazı yeni işlevleri olan yeni bir sınıf yazmaktır.
- Bu, çok eski bir iştir, özellikle de eski sınıfı iş üzerinde değiştirmeye karar vererseniz.
- Bunun için sınıf kalıtımı var.
- Sınıflar, diğer sınıflardan metot ve değişkenleri miras alabilir.
- A sınıfı B sınıfından miras alırsa, B sınıfı süper sınıf, A sınıfı ise alt sınıf olarak adlandırılır.
- Sınıflar, üst sınıftaki tüm yöntem ve değişkenleri devralır.
- Biri alt sınıfta uygun şekilde metotların üzerine yazma (overwrite) yapabilir veya yeni metotlar ekleyebilir.

```
class Class_name (Subclass_name) :  
    block
```

# Lambda fonksiyonlar

- lambda keyword'ü ile tanımlanan tek ifadeden oluşan bir fonksiyon
- ```
def short_func(x):  
    return x*2
```
- ```
short_func2 = lambda x: x*2
```
- ```
In [177]: strings = ['foo', 'card', 'bar', 'aaa', 'abab']  
In [178]: strings.sort(key=lambda x: len(set(list(x))))  
In [179]: strings  
Out [179]: ['aaa', 'foo', 'abab', 'bar', 'card']
```
- ```
In [180]: strings.sort(key=lambda x: len(set(list(x))), reverse=True)  
In [181]: strings  
Out [181]: ['card', 'bax', 'bar', 'foo', 'abab', 'aaa']
```



# map, filter, reduce

- `map( )` function basically executes the function that is defined to each of the list's element separately.

```
list1 = [1,2,3,4,5,6,7,8,9]
```

```
eg = map(lambda x:x+2, list1)
```

```
print(eg)
```

```
[3, 4, 5, 6, 7, 8, 9, 10, 11]
```

- `filter( )` function is used to filter out the values in a list. Note that `filter()` function returns the result in a new list.

```
liste2=filter(lambda x:x<5,list1)
```

```
print liste2
```

```
print list1
```

```
[1, 2, 3, 4]
```

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

- `reduce( )`

```
sayilar = range(1,5)
```

```
print sayilar
```

```
[1, 2, 3, 4]
```

```
reduce(lambda sayi1,sayi2:sayi1*sayi2,sayilar)
```

```
24
```

# enumerate, zip, sorted, reversed

- Sequence üzerindeki iterasyonlarda anlık indeksi tutmak için

```
i=0
```

```
for value in collection:
```

```
    #do something with value
```

```
    i+=1
```

```
for i, value in enumerate(collection):  
    #do something with value
```

- In [1]: some\_list=["foo", "bar", "baz"]  
In [2]: mapping={}  
In [3]: for i, v in enumerate (some\_list):  
 mapping[v]=i

```
In [4]: mapping
```

```
Out[4]: {'bar': 1, 'baz': 2, 'foo': 0}
```

- Herhangi bir sequence için sıralı liste doner.

```
In [5]: sorted([7, 1, 2, 6, 0, 3, 2])
```

```
Out[5]: {0, 1, 2, 2, 3, 6, 7}
```

# enumerate, zip, sorted, reversed

- zip; tuple listesi oluşturmak için Liste, tuple ve diğer sequence'lerin elemanlarını birleştirir.
- In [1]: some\_list=["foo", "bar", "baz"]  
In [2]: some\_list2=["one", "two", "three"]  
In [3]: zipped=zip(some\_list, some\_list2)  
In [4]: list(zipped)  
Out[4]: [("foo", "one"), ("bar", "two"), ("baz", "three")]
- reversed, sequence üzerinde ters sırada iterasyon sağlar.
- In [5]: list(reversed(range(10)))  
Out[5]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# Uygulamalar

- Piazza ders kaynaklar içindekilere bknz:  
01.ipynb, 02.ipynb, 03.ipynb, 04.ipynb, 05.ipynb,  
06.ipynb ve 07.ipynb
- Kod çalışırken neler olduğunu görsel yorumlamak  
için  
<http://pythontutor.com/>