Fonksiyonel Programlama

Dr. Yunus Santur

Dersin Amacı

- Bir bilgisayar programı dışarıdan bazı girdileri alır, onlar üzerinde işlem yapar ve bazı çıktılar üretir. Matematiksel bir gözle bakarsanız aslında bütün bir programın matematikteki bir fonksiyona benzediğini görürsünüz.
- ***program(girdiler) = çıktılar***

- **(Vikipedia) Fonksiyonel programlama**, yalnızca fonksiyonların kullanılmasıyla yazılmış programlardır. Fonksiyonel programların tipik özellikleri:
  - Atama deyimi bulunmaz. Değişkenlerin değeri bir kere verildi mi, bir daha değişmez.
  - Yan etkiler yoktur. Bir fonksiyonu çağırmak kendi sonucunu hesaplamaktan başka bir etki üretmez.

Ders İçeriği

- Bu ders kapsamında fonksiyonel programlama paradigmasını Python dili uygulamaları eşliğinde işleyeceğiz.

- Fonksiyonel Programlama Dilleri
  - Lisp
  - Haskell
  - Scala

Ders Planı

1	Giriş	Programlama paradigmaları, prosedüel programlama, nesne yönelimli, doğrusal programlama, dinamik programlama, fonksiyonel programlama, Diller, Python diline giriş
2	Python programlama dili	Python programlama, veri yapıları, döngüler, şart ifadeleri, nesne yönelimli programlama
3	Gömülü fonksiyonlar	Gömülü fonksiyonlar, fonksiyonlar, rekürsif fonksiyonlar
4	Fonksiyonel veri yapıları 1	Soyut veri yapıları, sözlük, küme, liste, demet, aralık
5	Fonksiyonel veri yapıları 2	Cebirsel veri yapıları
6	Fonksiyonel liste üreteçleri	Generatorler, liste üreteçleri
7	Python fonksiyonel programlama araçları 1	Lambda, Yield
8	Python fonksiyonel programlama araçları 2	Map, Filter, Reduce
9	Python fonksiyonel programlama araçları 3	Functool, itertool, decorators
10	Yüksek dereceli fonksiyonlar	Yüksek seviyeli foknsiyonlar, tembel değerlendirme
11	Monadslar	Monads
12	Projelerle fonksiyonel programlama projeler	Veri bilimine giriş, veri yükleme, veri manipülasyonu, vektörler, kütüphaneler: numpy, scipy, matplotlib
13	Projelerle fonksiyonel programlama projeler	Veri görselleştirme
14	Projelerle fonksiyonel programlama projeler	Metin işleme, görüntü işleme, zaman serileri

Programlama Paradigmaları

- Assembly
- Yüksek seviyeli diller
- Prosedürsel programlama
- Nesne tabanlı programlama
- Mvc
- Rapid development
- Extreme programming

Değerlendirme

- Vize (%40) + Final (%60)
- Aynı zamanda projeler verilecek, bu projeler vize ve final içinde ağırlıklandırılacak.

Python

- 1990 yılında geliştirildi
- C temelli
- Guido Van Rossum (Baş Geliştirici)
- Açık kaynak
- Cross-platfrom
- Öğrenmesi kolay: «Hayat kısa python öğrenin»
- Yaygın kullanım

Kimler Kullanıyor

- Gömülü Sistemler
  - Aurdino, micropython
- Gerçek Dünya Uygulamaları
  - Google sürücüsüz araba
- Bilim Dünyası
  - Tensorflow, keras, scipy, matplotlib
- Web
  - Instagram, Dropbox, Google
- Oyun/3B modelleme
  - Blender, Maya
- Standart Programlama Dili Olarak
  - Standford, Mit

Genel Özellikler

- Python hızlı öğrenme ve rapid geliştirme kurgusu üzerine kurulu bir dildir. Bu nedenle
  - Küme parantezi
  - Begin-end benzeri ifadeler
  - Satır sonu ;
  - Değişken tipi tanımlama yoktur.

  - Girintileme kuralı vardır
  - Belirli bir blok'a ait kodlar girintilime yapılarak yazılmalıdır

Python IDE

- Python için geliştirme ortamları
  - Anaconda (Bir çok kütüphane entegre olarak gelmektedir)
  - Pycharm (Anaconda kadar güçlü bir IDE)
  - Diğer editörler (Atom, Sublime, Idle)
- Online Ortamlar
  - https://repl.it/languages/python3

Anaconda Ortamı

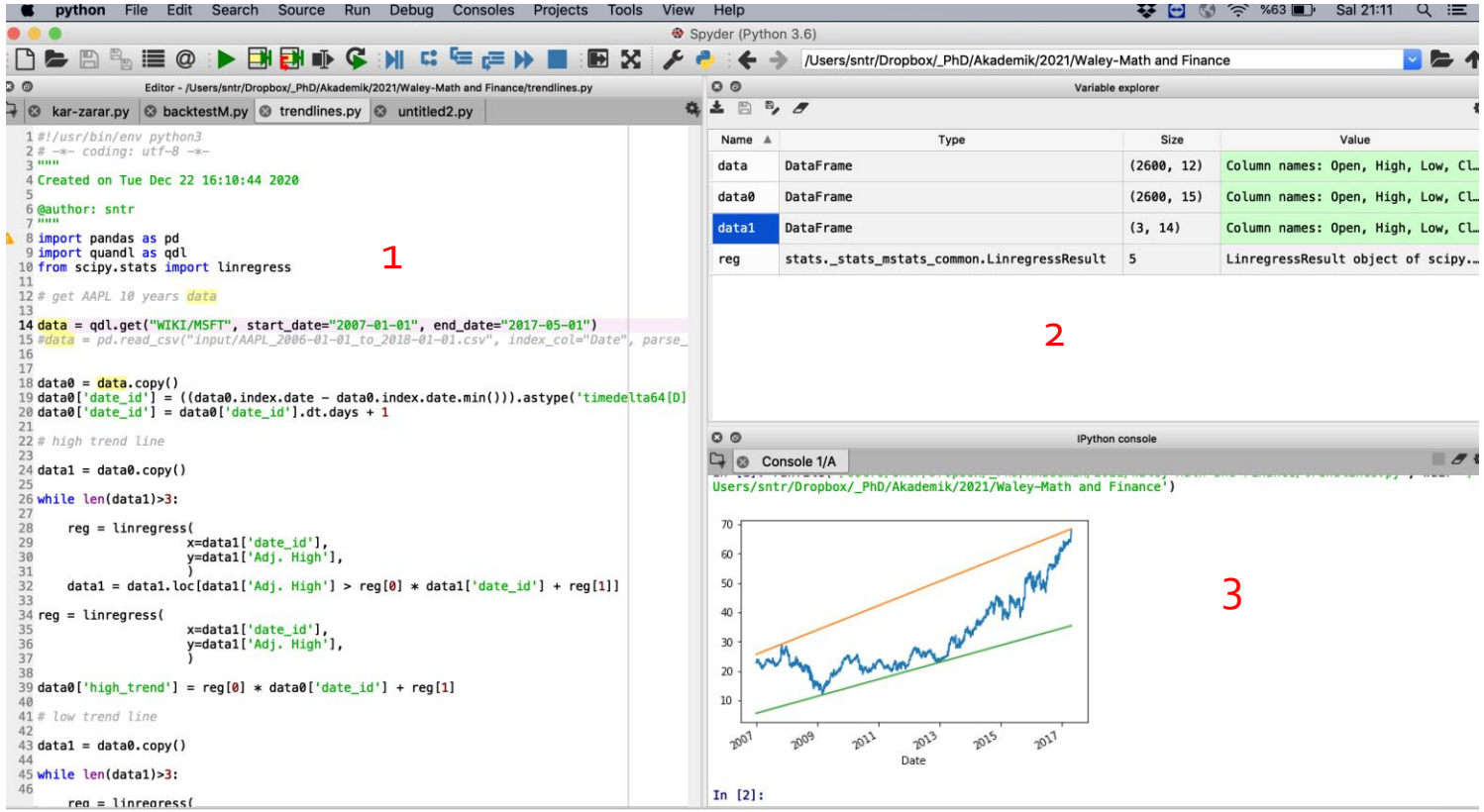
- Çok kullanılan kütüphanelerin yanısıra aşağıdakileri içerir.
  - JupyterLab : İnteraktif python
  - Jupyter: Web tabanlı interaktif python
  - Rstudio: R
  - Orange: Veri madenciliği kütüphanesi
  - Spyder: Editör

Anaconda ortamı

1: Kod ortamı

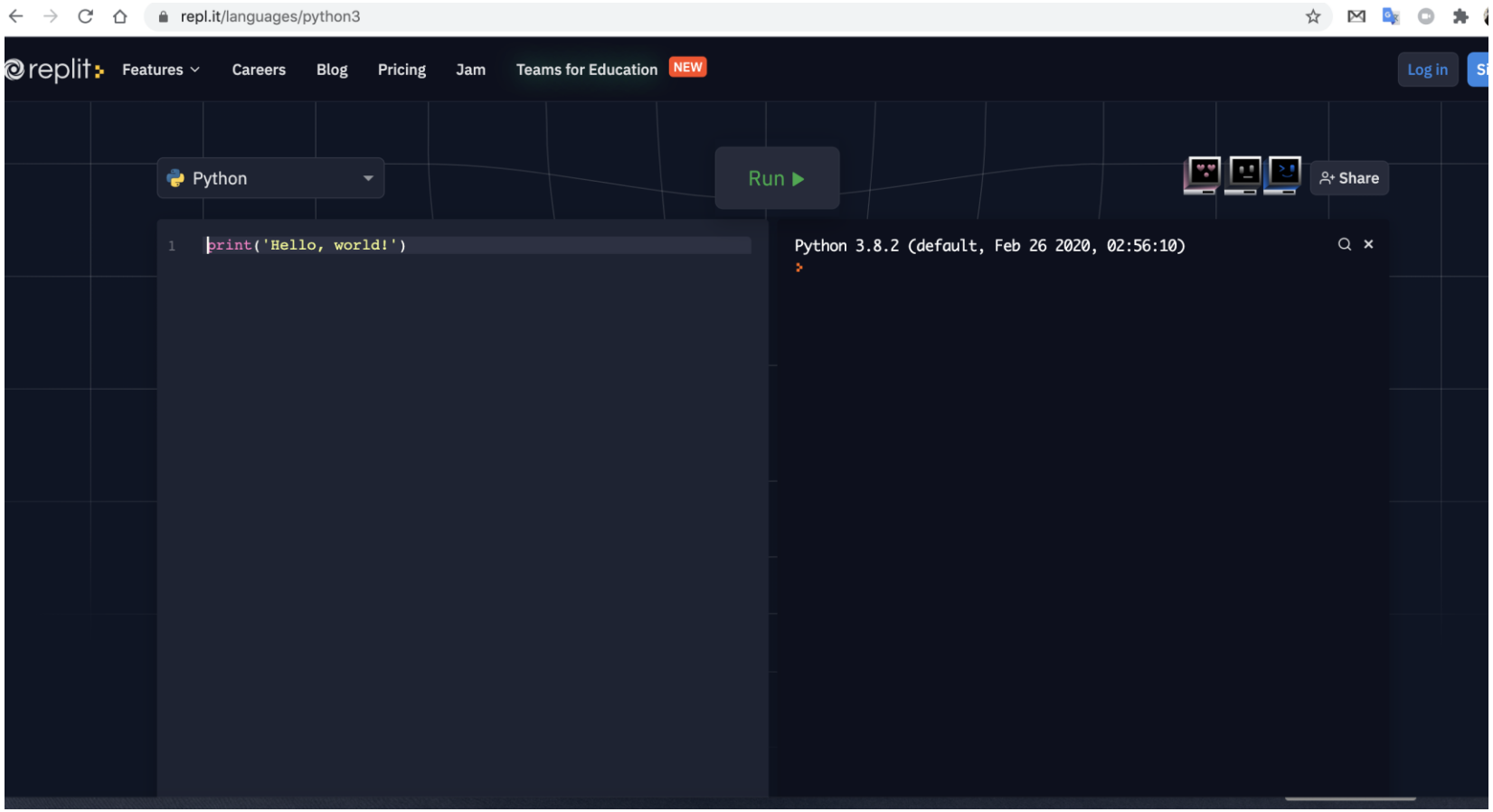
2: Explorer

3: Konsol



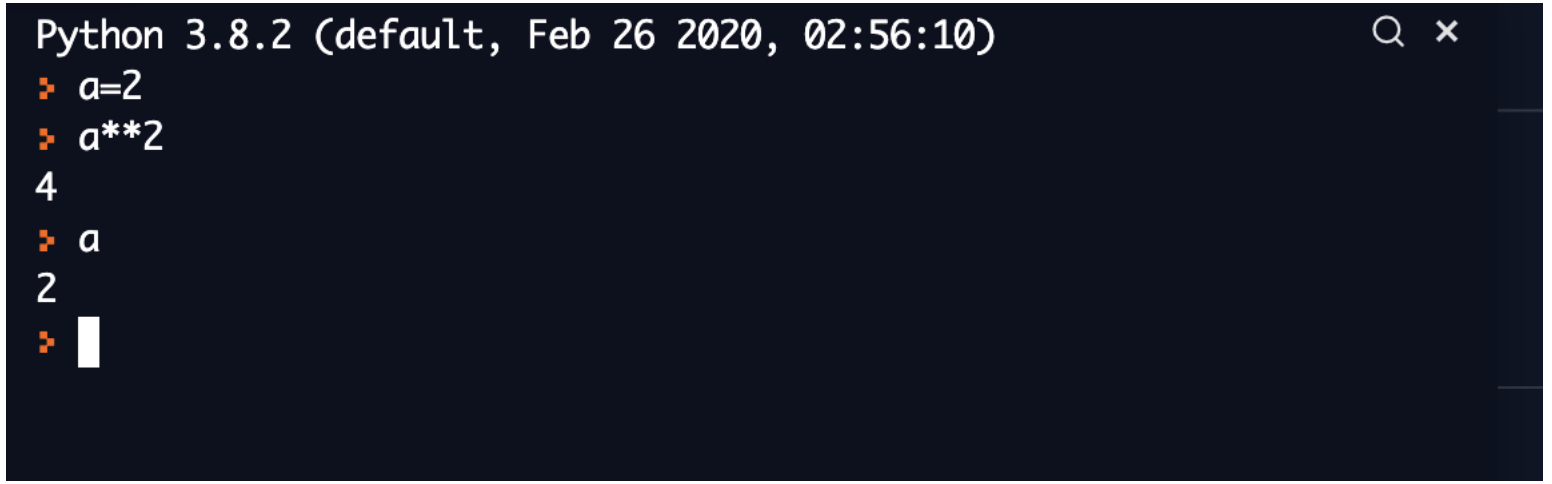
repl.it ortamı

- Sol taraf editör, sağ taraf ise hem konsol hem etkileşimli python kabuğu



Python öğrenmeye başlamak

- Etkileşimli kabuk örnekleri



Python öğrenmeye başlamak

- Değişken tipi tanımlamaya gerek yoktur. Peki değişkenimiz ne olarak tanındı ?

type(değişken_adı)

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10)
> a=2
> type(a)
<class 'int'>
>
```

Değişken tanımlamadan da yapılabilir.

```
> type("yazilim muhendisligi")
<class 'str'>
>
```

Python öğrenmeye başlamak

- Örneğin string (metin) ifade ile ilgili hangi fonksiyonları kullanabiliriz? Komutumuz **dir(değişken_adi)**
- Burada yer alan fonksiyonlar aynı zamanda gömülü fonksiyon olarak isimlendirilir. Yani bunları kullanmak için ekstra bir kütüphane yüklenmesine gerek yoktur.

Python öğrenmeye başlamak

- Hangi fonksiyonları kullanacağımızı dir metotu ile gördük, peki nasıl kullanacağız. Komutumuzun adı **help**
- Kullanılışı **help(değişken_adi.gömülü_fonksiyon_ismi)**
- Bu örnekte swapcase gömülü fonksiyonunun ne işe yaradığını varsa diğer giriş ve çıkış değerleri ile açıklamaları görüyoruz.

```
: help(x.swapcase)
Help on built-in function swapcase:

swapcase() method of builtins.str instance
    Convert uppercase characters to lowercase and lowercase char
acters to uppercase.
```

Python öğrenmeye başlamak

- Şu ana kadar geliştirme ortamlarını ve aşağıdaki 3 komutu öğrendik.
  - **type**
  - **dir**
  - **help**
- Özellikle **dir()** ve **help()** kod yazarken en büyük yardımcılarınız olmalı. Kod yazarken unuttuğumuz yerde kitap/web zaman kaybetmek yerine önce konsolda bu komutlardan yararlanacağız.
- Bu bize geliştirme yaparken hız kazandıracak bir alışkanlık olacak.

Ekrana yazdırma

- **print()** ifadesini kullanıyoruz.
- Python programlama dili iki ana sürüme sahiptir.
  - Python 2, print ifadesinde parantez kullanımı seçimliktir.
  - Python 3, print ifadesinde parantez zorunludur.

Ekrana yazdırma

- Sayısal ve metin ifadeleri birlikte kullanmak. Çift tırnak içinde yer alan tüm ifadeler doğrudan ekrana yazdırılır.

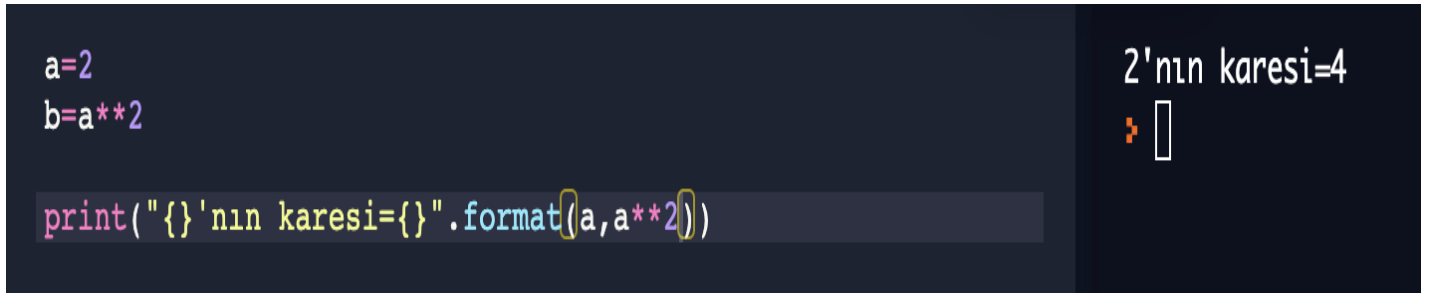
```
a=2
b=a**2

print(a,"'nın karesi=",a**2)
```

```
2 'nın karesi= 4
>
```

Ekrana yazdırma

- Bir önceki ile aynı işi yapar buna alternatif olarak **format()** fonksiyonunu da kullanabiliriz.
- Bu kullanımda tüm metin ifadelerini çift tırnak içinde yazarken, araya yerleşecek her değişken için {} (küme parantezleri) kullanıyoruz ve çift tırnaktan sonra .format(var1, var2 …) şeklinde sırası ile yazıyoruz.



Format kullanımı

- Bir diğer kullanım şekli kesirli sayıların yazdırılması. Bu örnekte virgülden sonra sadece 2 hane yazılmasını sağladık.

```
a=2/3

print("a=",a)

print("a={:.2f}".format(a))
```

```
a= 0.6666666666666666
a=0.67
```

Format kullanımı

- Format için bir başka örnek: parantezler içinde yazılacak rakamları format içinde yazılan değerlerin indisi olarak değerlendirilir.

```
print("{1} {2} {0}".format(3,1,2))
```

```
1 2 3
```

İlk program

- Döngüler konusu ayrıca işlenecek. Burada **girintileme** ve **:** kullanımına dikkat çekilmek isteniyor. Blok (döngü, şart, metot gövdesi) ifadelerinden sonra parantez blokları kullanılmıyor. Bloğa ait ifadeler girintileme yapılarak yazılıyor. Editör ortamında girintileme için tab tuşu kullanmak yeterli.

Açıklama satırları

- Tek satırlık açıklamalar için **#**
- Çok satırlı açıklamalar için çok satırlı metini **3 er adet çift tırnak** işareti içine alıyoruz.

```
1   #tek satırlık açıklama
2
3   """
4   aşağıdaki kod 0'dan 10'a kadar
5   2'şer artımlı bir döngü kurarak
6   bu sayıları ekrana yazdırır
7   """
8   for i in range(0,10,2):
9       print(i)
10
11
```

Python Programlama Dili

Dr. Yunus Santur

Aritmetik İşleçler

+	Toplama
-	Çıkarma
*	Çarpma
**	Üst alma
/	Bölme
//	Tam sayı bölme
%	Bölmeden kalan
+=	Topla ve eşitle
-=	Çıkar ve eşitle
*=	Çarp ve eşitle
/=	Böl ve eşitle

Aritmetik işleçler örnek

```
1  a=9
2  b=a//4
3  c=a%4
4  print(b,c)
```

```
2 1
```

Karşılaştırma İşleçleri

- == Eşit ise
- != Eşit değil ise
- > Büyük ise
- < Küçük ise
- >= Büyük veya eşit ise
- <= Küçük veya eşit ise

Bool işleçler

- and
- or
- not
- in

```
1  x="p" in "Python"
2  print(x)
3
4  y= 5 in [3,5,7]
5  print(y)
6
```

Şart ifadeleri

if *boolean_ifade*:

    if şartına ait kodlar

if boolean_ifade:

        şart ifadesi doğru ise

else:

        şart ifadesi yanlış ise

Şart ifadeleri

```
if şart:
        şart1 doğru ise
elif şart1:
        şart2 doğru ise
elif şart2:
        şart3 doğru ise

...

else:
        default kısım
```

Şart ifadeleri örnek

- Alıştırma: Eşkenar üçgen problemini çözmeye çalışın

```
a = 200
b = 33
if b > a:
  print("b büyük a")
elif a == b:
  print("eşitler")
else:
  print("a büyük b")
```

Kısa if

- Python sözdizimi gereği her satıra bir kod/komut gelmelidir. Ancak bazı kısaltma ifadeleri girintileme gibi kuralları bozmadan kısa kodlar yazmamızı sağlar.
- Aşağıdaki örnek else kısmı olmayan bir if örneği ve : kullandık

```
a = 200
b = 33
if a>b: print("a büyük b")
```

Kısa if

- Aşağıdaki örnekte şart doğru ise kısmı iften önce, şart yanlış ise kısmı iften sonra yazılmıştır ve : kullanılmamıştır.

1 if True else 0

```
1 if True else 0
1
```

```
a = 200
b = 33
print("a büyük b") if a>b else print("b büyük")
```

Veri tipleri

- Temel tipler (integer, string, boolean,double)

- Diğer veri tipleri
  - Liste (List)
  - Demet (Tuple)
  - Küme (Set)
  - Sözlük (Dictionary)

Listeler

```
a=[1,2,"a",3,5]       # Dizilere benzerler ancak boyutları dinamiktir, ve karışık türden veriler içerebilirler.

x=list()              # Boş liste oluşturur
x=[]                  # Boş liste oluşturur


len(a)                # Eleman sayısı
a.sort()              # listeyi sıralar
a.reverse()           # listeyi ters çevirir
a.pop()               # son elemanı siler
a.append("a")         # sonuna yeni eleman ekler
a.insert(indis, "a")  # yeni elemanı belirtilen indise ekler
```

Listeler devam

```
a.count(1)      # Bu eleman listede kaç tane var
a.index(1)      # Bu eleman kaçıncı indiste

print a[1]      # 1.indiste ki elemanı yazdır
a[1]=2          # 1.indisteki elemanın değerini değiştir
del a[2]        # 2.indisteki elemanı listeden sil
```

Listeler yardım

Yardım ?

```
>>> dir(a)            # a ile başka ne yapabilirim
```

Kabukta belirli bir fonksiyonla ilgili yardım alma

```
>>> help(a.append)      # a.append nasıl çalışır
```

Listeler örnek

- Listelerler örnek

```
> x=[1,2,3]
> len(x)
3
> x.append(7)
> x
[1, 2, 3, 7]
> len(x)
4
> x.pop()
7
> x
[1, 2, 3]
> x.insert(0,-1)
> x
[-1, 1, 2, 3]
> del x[0]
> x
[1, 2, 3]
> x[-1]
3
```

Döngüler

- for
- while

- Python dilinde döngüler daha esnektir ve listeler ile ileride göreceğimiz sözlük, demet, küme gibi veri yapıları itere edilebilir yada diğer bir ifade ile mutable nesnelerdir.

- Yapısı gereği, for döngüleri daha sık tercih edilir.

For döngüsü

for ***döngü_değişkeni*** in ***obje***:
    Döngüye ait blok komutları

Aşağıdaki örnekte i döngü değişkeni (sırası ile x listesindeki elemanların yerine geçmiştir) x ise liste olduğu için mutable bir nesnedir. Bir sonraki örnekte x elemanlarını toplayalım.

```
x=[1,2,3,4,5]

for i in x:
  print(i)
```

```
1
2
3
4
5
```

For döngüsü örnek

- İlk örnekte döngü kurarak liste içindeki elemanların toplamı bulunmuştur.
- İkinci örnekte gösterilen sum() gömülü fonksiyonu bu işi pratik yapar.

```
x=[1,2,3,4,5]
t=0
for i in x:
  t+=i

print(t)
print(sum(x))
```

Range komutu

- Gerek döngülerde, gerek bir sayı aralığı oluşturmak için kullanılır. 1-3 parametre alır, 2 ve 3. parametreler seçimliktir.
- Tek parametre ile kullanıldığında
  - range(5) => 0, 1, 2, 3, 4 #Başlangıç değeri varsayılan 0 oldu.
  - range(2,5) =>2,3,4 #Başlangıç değeri 2 oldu
  - range(1,10,2) => 1, 3, 5, 7, 9 #Başlangıç:1, **adım:**2

  - Dikkat edelim son sayılar aralığa dahil edilmez. Aralıklar varsayılan olarak küçükten büyüğe kurgulanır. Ancak aşağıdaki de doğrudur.
  - range(10,5,-1) => 10, 9, 8, 7, 6

Range örnekler

- range öncesinde bir liste değişkenine atanabileceği gibi doğrudan döngü nesnesi olarak kullanılabilir. İkinci kullanım türü Java benzeri döngü yapılarına daha çok benzemektedir.

```
x=range(10,5,-1)
for i in x:
  print(i)
```

```
for i in range(0,100,2):
  print(i)
```

break ve contine kullanımı

- Döngülerle birlikte
  - Break: Döngüyü bitirir. İç içe döngüler var ise içteki döngüyü bitirir.
  - Continue: Pas geç

```
for i in range(1,10):
  if i%2==0:
    continue
  elif i%5==0:
    break
  else:
    print(i)
```

break ve contine kullanımı

- Çift sayılar pas geçilir ve 5'e geldiğinde döngü sonlandırılır.

```
for i in range(1,10):
  if i%2==0:
    continue
  elif i%5==0:
    break
  else:
    print(i)
```

```
1
3
```

For döngüsü örnek:1 – asal sayı

- Asal sayı örneği
  - 1 ve kendisinden başka böleni olmayacak.
  - asal bizim konrtrol (bayrak) değişkenimiz. Döngü içinde herhangibir sayıya bölünürse değerini False yapıp döngüden çıkıyoruz.
  - Döngü sonunda değişkenin değerini tekrar kontrol ediyoruz.

```
asal=True
x=17

for i in range(3,x):
  if x%i==0:
    asal=False
    break


print(x," asaldir") if asal else print(x," asal degildir")
```

For - else örnek:2 – asal sayı

- Asal sayı örneği: Pythonic versiyon
  - ***Döngülerle birlikte else kullanabiliyoruz***
  - Daha az kod, bayrak değişkenine gerek yok
  - Else kısmı döngü sonunda çalışır. Daha öncesinde break çalışır ise else kısmı çalışmamış olur.

```
x=18

for i in range(3,x):
  if x%i==0:
    print(x, "asal değildir")
    break
else:
  print(x," asaldir")
```

For else döngüsü

- Aşağıdaki iki örnek arasında fark yok. Else kullanımı burada ekstra bir niteliğe sahip değil.

```
x=10
t=0
for i in range(x):
  t+=i
else:
  print(" toplam=",t)
```

```
x=10
t=0
for i in range(x):
  t+=i

print(" toplam=",t)
```

While döngüsü

- For döngüleri daha esnektir. While döngülerinde, döngü değişkeni ve döngü içinde bu değişkenin güncellenmesine ihtiyaç vardır.

```
x=10
t=0
i=0
while i<x:
  t+=i
  i+=1


print(" toplam=",t)
```

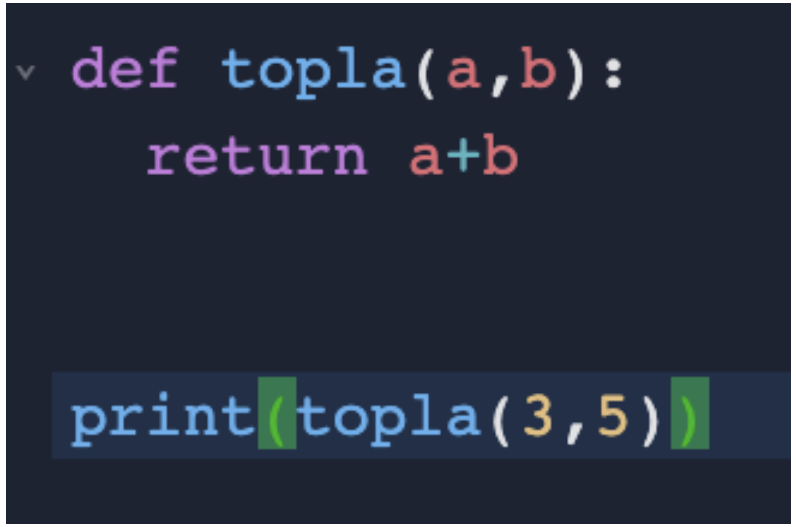
While-else döngüsü

- Asal örneğini while-else döngüsü ile tekrarlayalım. Unutmayalım else döngü sonunda çalışacak, döngü daha erken biterse (yani sayi asal değilse) else kısmı çalışmamış olacak.

```
x=17
i=2
while i<x:
  if x%i==0:
    print(x," asal değil")
    break
  i+=1
else:
  print(x," asaldir")
```

Metotlar

- def anahtar sözcüğü ile başlar, return ile sonucu geri dönderirler.



Metotlar

- def anahtar sözcüğü ile başlar, return ile sonucu geri dönderirler.
- Parametrelerin default değerleri parametre listesinde verilebilir.
- N parametre varsa o-n arasında kullanılabilir.
- Parametreler soldan sağa sıralı olarak geçer.

```
def topla(a=3,b=5):
  return a+b


print(topla())
```

```
def topla(a=3,b=5):
  return a+b


print(topla(5))
```

Sınıflar

- class anahtar sözcüğü ile oluşturulur, büyük harf zorunludur.
- __init__ metotu yapılandırıcı metottur.
- Self anahtar sözcüğü kullanımı zorunludur. Java dilindeki this kullanımına benzer, sınıfa atıf yapar.

```
class Matematik:
  def __init__(self):
    self.pi=3.14

  def alan_hesapla(self,r=0):
    return self.pi* (r**2)

  def cevre_hesapla(self,r=0):
    return self.pi*2*r



daire=Matematik()

print (daire.alan_hesapla(4))

print (daire.cevre_hesapla(4))
```

Python Programlama Dili

Gömülü Fonksiyonlar, Rekürsif fonksiyonlar

Yunus Santur

Time kütüphanesi

- Kütüphaneleri eklemek için **import *kütüphane_ismi***

```
> import time
> dir(time)
['CLOCK_BOOTTIME', 'CLOCK_MONOTONIC', 'CLOCK_MONOTONIC_RAW', 'CL
OCK_PROCESS_CPUTIME_ID', 'CLOCK_REALTIME', 'CLOCK_THREAD_CPUTIME
_ID', '_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '
__package__', '__spec__', 'altzone', 'asctime', 'clock_getres',
'clock_gettime', 'clock_gettime_ns', 'clock_settime', 'clock_set
time_ns', 'ctime', 'daylight', 'get_clock_info', 'gmtime', 'loca
ltime', 'mktime', 'monotonic', 'monotonic_ns', 'perf_counter', '
perf_counter_ns', 'process_time', 'process_time_ns', 'pthread_ge
tcpuclockid', 'sleep', 'strftime', 'strptime', 'struct_time', 't
hread_time', 'thread_time_ns', 'time', 'time_ns', 'timezone', 't
zname', 'tzset']
>
```

Time kütüphanesi

time.gmtime()

Zamanı verir, diğer parametrelerine . Koyarak erişebiliriz.

```
> import time
> time.gmtime()
time.struct_time(tm_year=2021, tm_mon=3, tm_mday=10, tm_hour=19,
 tm_min=31, tm_sec=3, tm_wday=2, tm_yday=69, tm_isdst=0)
>
> time.gmtime().tm_mday
10
```

time.time()

Bailangıç zamanından, şu ana kadar geçen süreyi verir

Başlangıç zamanı ney? **time.gmtime(o)**

Zamanı manipüle etmek

- Aşağıdaki programın ne işe yarayacağını düşünelim

```
import time

t1=time.time()
t1=t1+60*60

t2=time.gmtime(t1)

print(t2)
```

Zamana formatlı yazmak

- time.asctime(time.localtime())

- time.strftime('%c')

%a: hafta gününün kısaltılmış adı
%A: hafta gününün tam adı
%b: ayın kısaltılmış adı
%B: ayın tam adı
%c: tam tarih, saat ve zaman bilgisi
%d: sayı değerli bir karakter dizisi olarak gün
%j: belli bir tarihin, yılın kaçıncı gününe denk geldiğini gösteren 1-366 arası bir sayı
%m: sayı değerli bir karakter dizisi olarak ay
%U: belli bir tarihin yılın kaçıncı haftasına geldiğini gösteren 0-53 arası bir sayı
%y: yılın son iki rakamı
%Y: yılın dört haneli tam hali
%x: tam tarih bilgisi
%X: tam saat bilgisi

time.slep()

- Parametre olarak saniye cinsinden değer alır, programın duraklatılmasını sağlar, gömülü sistemlerde işe yarayabilir.

```
import time
for i in range(5):
    print(i)
    time.sleep(0.5)
```

Random kütüphanesi

- Varsayılan olarak bu komut 0 ile 1 arasında rastgele bir sayı üretir. 1 dahil değildir.
- Örnek: Sadece bu fonksiyonu kullanarak istenilen a-b aralığında tam sayı üretmeye çalışalım?

```
import random

print(random.random())
```

Random kütüphanesi

- Random kütüphanesinde tam sayı üretme ile ilgili fonksiyonlar mevcuttur ancak biraz kodlama ile başka fonksiyon öğrenmeye gerek kalmadan yapmak mümkündür. (int) kullanımı tip dönüşümü yapmaktadır.
- Pratik olarak **random.randint(a,b)** aynı işi yapar.

```
import random

# 5-10 arasında sayı üretmeye çalışıcaz
a=5
b=10

x=a+ (int)((b-a)*random.random())

print(x)
```

Random kütüphanesi

- Diğer fonksiyonlar için dir(random) diyerek yardım alabiliriz.

```
> import random
> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGI
CCONST', 'SystemRandom', 'TWOPI', '_Sequence', '_Set', '__all__'
, '__builtins__', '__cached__', '__doc__', '__file__', '__loader
__', '__name__', '__package__', '__spec__', '_accumulate', '_aco
s', '_bisect', '_ceil', '_cos', '_e', '_exp', '_inst', '_log', '
_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt', '
_test', '_test_generator', '_urandom', '_warn', 'betavariate', '
choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'get
randbits', 'getstate', 'lognormvariate', 'normalvariate', 'paret
ovariate', 'randint', 'random', 'randrange', 'sample', 'seed', '
setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate'
, 'weibullvariate']
>
```

Random faydalı fonksiyonlar

- Shuffle , choice, sample
- Choice listeden tek bir elemanı rastgele seçer.
- Sample ise parametre sayısı olarak verilen kadar rastgele eleman seçer.
- Shuffle elemanları karıştırır. Örneğin bir sınav otomasyonunda soruların herkese karışık sırada getirilmesi için kullanılabilir.

```
import random

x=[1,2,3,'a','b']

print (random.choice(x))

print (random.sample(x,2))
```

```
import random

x=[1,2,3,'a','b']

random.shuffle(x)
print(x)
```

String işlemleri

- Her string bir karakter dizisidir ve mutable nesnelerdir. Dolayısı ile doğrudan itere edilebilirler. İndislerin o dan başladığını unutmayalım.

```
x="Python programlama dili"

for i in x:
  print(i)

print(x[4])
```

String işlemleri

- Stringleri bölmek. Stringlerden (Listeler içinde aynı şey geçerlidir) : ile belirli aralıkları parçalayabiliriz.

```
x="Python programlama dili"

print(x[2:10]) #2'den 10.indise kadar

print(x[2:]) #2'den sonuna kadar

print(x[:10]) #Baştan 10.indise kadar

print(x[0::2]) #Baştan sona ikişer adımla

print(x[::-1]) #Sondan başa -1 adıma, ters çevirmiş olduk
```

String manipülasyonu

- Büyük/küçük dönüşümü, boşluk veya özel karakter kaldırma, değiştirme, bölme (split) ve birleştirme (join)

```
> x=" Python programlama, dili "
>
> x.strip(' ')
'Python programlama, dili'
>
> x.replace(',',' ')
' Python programlama  dili '
>
> x.upper()
' PYTHON PROGRAMLAMA, DILI '
>
> x.lower()
' python programlama, dili '
>
> a=x.split(' ')
> a
['', 'Python', 'programlama,', 'dili', '']
```

```
> ':'.join(a)
':Python:programlama,:dili:'
```

Math modülü

- Bazı sık kullanılabilecek fonksiyonlar

```
> import math
>
> math.pow(4,3)
64.0
>
> min(3,4,5)
3
> math.ceil(2.01)
3
> math.floor(2.01)
2
> math.sqrt(64)
8.0
> math.pi
3.141592653589793
>
> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ce
il', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fm
od', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma', 'log'
, 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow',
'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 't
anh', 'tau', 'trunc']
```

Locale modülü

- Dil ve tarih gibi bazı nesnelerin yerelleştirilmesini sağlar. Bu örneği Jupyter labta yapmış olduk.

```
In [1]: import time
        time.strftime("%B")

Out[1]: 'March'

In [2]: import locale
        locale.setlocale(locale.LC_ALL, "tr_TR")
        time.strftime("%B")

Out[2]: 'Mart'
```

Kullanıcıdan girdi alma

- input() komutu ile alınır ancak girdiler string olarak yorumlanır sayısal işlemlerde kullanmak için tip dönüşümüne ihtiyaç vardır.

Ödev ve alıştırmalar

- Aşağıdaki alıştırmaları bir dahaki derse kadar yapalım

1. En az bir büyük harf, en az bir küçük harf, en az 8 karakter alfanumerik bir şifre oluşturma politikasına göre rastgele şifre üreten bir program yazınız.
2. Rastgele şifreler belirleyerek yada üreterek 1.maddede ki kriterlere uymayan şifreleri belirleyiniz
3. Konsoldan oynanan bir sayısal-loto programı geliştirin. Programın tuttuğu ile bizim girdiğimiz sayıdan kaç tanesi tutacak.
4. Konsoldan oynanacak adam-asmaca oyunu geliştiriniz. Kelime havuzunu liste olarak tanımlayıp başlangıçta uzunluğa göre 3-4 harfi rastgele kullanıcıya vererek gerisini kullanıcının girmesini sağlayın.

Python Programlama Dili

Fonksiyonel veri yapıları

Yunus Santur

Göreceğimiz veri yapıları

- Listeler (**list**) []
  - Dinamik
- Demetler (**tupple**) ()
  - Güncellenemezler
- Kümeler (**set**){}
  - Tekrar eden eğerler içermezler
- Sözlükler (**dict**){key:value}

- **Hepsi mutable nesnelerdir.**

Listeler

Listeler

```
a=[1,2,"a
",3,5]      # Veriler karışık türden olabilir

len(a)                  # Eleman sayısı
a.sort()                # listeyi sıralar
a.reverse()             # listeyi ters çevirir
a.pop()                 # son elemanı siler
a.append("a")           # sonuna yeni eleman ekler
a.insert(indis, "a")  # yeni elemanı belirtilen indise ekler
```

Listeler devam

```
a.count(1)      # Bu eleman listede kaç tane var
a.index(1)      # Bu eleman kaçıncı indiste

print a[1]      # 1.indiste ki elemanı yazdır
a[1]=2          # 1.indisteki elemanın değerini değiştir
del a[2]        # 2.indisteki elemanı listeden sil

x=list()        # Boş liste oluşturur
x=[]            # Boş liste oluşturur
```

Liste üzerinde döngüler

```
liste=["a","b","c"]
for i in liste:
        print(i)
```

Listeler enumerate

Soru: Döngülerde indislere de erişmek istersek ?

Cevap: enumerate

```
x=[1,2,3,4,5]

for i in enumerate(x):
  print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
```

```
x=[1,2,3,4,5]

for indis, eleman in enumerate(x):
  print(indis, eleman)
```

```
0 1
1 2
2 3
3 4
4 5
```

Listeler enumerate

Enumerate

```
x=[1,2,3,4,5]

for indis, eleman in enumerate(x):
  print("x[{}]={}".format(indis,eleman))
```

```
x[0]=1
x[1]=2
x[2]=3
x[3]=4
x[4]=5
```

Listeleri Birleştirmek

```
a=[1,2,3]
b=[4,5,6]
c=a+b
```

Çok boyutlu listeler

```
x=[
    [1,0,0],
    [0,1,0],
    [0,0,1]
  ]

len(x)      # satır sayısını verir
len(x[0])   # ilk satırdaki sütun sayını verir
```

Çok boyutlu listeler

```
import random
x=[]
for i in range(5):
    x.append([random.randint(1,5) for c in range(4)])


print (x)
```

```
[[3, 3, 1, 1], [3, 1, 5, 3], [4, 4, 4, 4], [5, 5, 2, 5], [1, 2
1, 2]]
```

Listeler-bölme

- String işlemlerinde görmüştük, negatif sayılar da kullanılabilir x[a:b:c]
  - a: başlangıç
  - b: bitiş
  - c: adım

```
> x=[1,2,3,4,5]
>
> x[0:4:2]
[1, 3]
>
> x[-1:-5:-2]
[5, 3]
>
> x[0:3]
[1, 2, 3]
>
> x[0:3]=[3,5,8]
> x
[3, 5, 8, 4, 5]
> sum(x)
25
>
> sort(x)
```

```
> x.sort()
> x
[3, 4, 5, 5, 8]
> x.reverse()
> x
[8, 5, 5, 4, 3]
```

Listeler-diğer

- Listeler veri yapıları dersinde geçen bağlı listeler olarak kullanılabilir.
- Boyutları dinamiktir, karışık veri türleri içerebilirler.

```
> x=list()
> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__delit
em__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '
__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd_
_', '__imul__', '__init__', '__init_subclass__', '__iter__', '__
le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__r
educe__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__
', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__sub
classhook__', 'append', 'clear', 'copy', 'count', 'extend', 'ind
ex', 'insert', 'pop', 'remove', 'reverse', 'sort']
>
```

Demetler

- Listeler gibi kullanılırlar ama güncellenemezler.

```
> x=()
> type(x)
<class 'tuple'>
>
> x=("veli","ali",1)
> x
('veli', 'ali', 1)
> x[0]
'veli'
> x[0]="fatma"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
> x.append("hakan")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
> len(x)
3
> x.count("ali")
1
> x.index("ali")
1
```

Demetler-güncelleme

- Doğrudan güncellenemezler

```
"""
Değiştirilemez demetleri nasıl değiştirebiliriz?
listeye dönüştürüo değiştirdik,
tekrar demet yapıp yeniden atama yaptık
"""
x=("ali","veli","hakan")

y=list(x)

y[1]="fatma"

x=tuple(y)

print(x)
```

Çoklu atamalar

- Hemen her veri tipinde kullanılabilir. Metotlardan dönüş değerlerinin alınmasında pratik kullanım sağlar.

```
1  gizliler=("ali","123")
2
3  user,sifre=gizliler
4
5  print(user,sifre)
```

```
ali 123
```

Kümeler

- İndeksleri yoktur, çift değer içermezler.

```
> x={1,2,3,2,1}
> x
{1, 2, 3}
> len(x)
3
>
> x[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>
> 1 in x
True
```

```
> x.update({1,2,3,4})
> x
{1, 2, 3, 4}
>
```

Sözlükler

- Anahtar-değer şeklinde veri yapılarıdır. Kullanım alanları oldukça fazladır.

```
x={"user":"ali","sifre":"123","login_count":5}

#Erişim
print(x['user'])

#update
x["user"]="ali cehver"

#yeni key-value ilave etme
x["tckimlik"]="12345566"

#key-value çifti silme
x.pop("tckimlik")

#sözlükteki eleman sayısı
len(x)
```

Sözlükler üzerinde döngüler

- Doğrudan itere edilebilirler.

```
x={"user":"ali","sifre":"123","login_count":5}


for k,v in x.items():
  print(k,":",v)
```

```
x={"user":"ali","sifre":"123","login_count":5}


for k in x.keys():
  print(k)
```

```
x={"user":"ali","sifre":"123","login_count":5}


for v in x.values():
  print(v)
```

Sözlüklere erişim

- Alternatif erişim yapıları oluşturabiliriz.

```
x={"user":"ali","sifre":"123","login_count":5}

u, p, lc=x.values()

print(u,p,lc)
```

Sözlüklere erişim

- Alternatif erişimler oluşturabiliriz.

```
x={"user":"ali","sifre":"123","login_count":5}

a=list(x.values())

print(a[0])
```

Listeler ve range

- Daha önce gördüğümüz range kullanarak istenilen sayı dizilerini oluşturmak mümkündür.

```
> x=range(1,10,2)
> type(x)
<class 'range'>
>
> x=list(range(1,10,2))
> type(x)
<class 'list'>
>
> x=[range(1,10,2)]
> type(x)
<class 'list'>
```

Python Programlama Dili

Liste Üreticiler

Yunus Santur

Üreticiler

- Bu bölümde liste, demet vb. oluşturmak için daha pythonic örnekler göreceğiz.

```
> range(1,10,2)
range(1, 10, 2)
> list(range(1,10,2))
[1, 3, 5, 7, 9]
```

```
> tuple(range(1,10,2))
(1, 3, 5, 7, 9)
> set(range(1,10,2))
{1, 3, 5, 7, 9}
>
```

Liste üreticiler

- Bu kullanımda y için: x'in içindeki her i ile y listesi oluştur, her i**2 ile z listesi oluştur demiş oluyoruz.

```
x=list(range(1,5))

y=[i for i in x]

z=[i**2 for i in x]
```

Liste üreticiler

- Bu örnek için z tahmini alalım?

```
z=[[i,i**2] for i in x]
```

Sözlük üreticiler

- Aynı şekilde sözlük üretelim. İç içe döngüler kullanmak yerine , Python gerçekten kolaylaştırıyor.

```
x=list(range(1,5))

z=[{str(i):i} for i in x]
```

Üreticilere şart ekelemek

- Listeden sayısal olmayanları eleyelim. Not: is ile başlayan metotların kullanımı unutmayalım.

```
x=["a",1,2,3,"b","c",4,5,6]

y=[i for i in x if str(i).isdigit()]
```

Örnek

- Örnek: Ortalamadan %20'den daha uzak olanları seçen bir liste üretici yazın. (Not: İlerleyen haftalarda bu amaçla farklı yapılar işlenecektir.)

Örnek: Çözüm

- Bu çözümden farklı olarak ne yapabiliriz ?

```
1  x=[0,1,2,3,4,5,6,7,8,9,10]
2  yuzde=20
3
4  y=[i for i in x if ( i> ((yuzde+100)*sum(x)) / (len(x)*100)
5    or i< ((100-yuzde)*sum(x)) / (len(x)*100) )]
6
```

Örnek

- İki liste üzerinde çift döngü örneği:

```
x=[1,2,3]
y=["a","b","c"]

z=[ [i,k] for i in x for k in y]
```

```
z
[[1, 'a'], [1, 'b'], [1, 'c'], [2, 'a'], [2, 'b'], [2, 'c'], [3,
 'a'], [3, 'b'], [3, 'c']]
```

Örnek

- Kullanıcı adlarında istemediğimiz karakter olan kullanıcılar listesi.

```
sifreler=["ali123.", "veli!12","kalem","kelam","kamil","gel.123"]

x=[i for i in sifreler if "." in i]
```

```
> x
['ali123.', 'gel.123']
>
```

Liste üreticileri ile kesişim, fark kümeleri oluşturmak

- Liste üreticileri kullanılarak kesişim, birleşim, fark kümeleri oluşturmak. (Aynı dersi alan öğrenciler gibi)

```
x=["python", "java", "perl", "scala"]

y=["ruby", "python", "java","php"]

kesisim=[i for i in x if (i in x and i in y)]

fark=[i for i in x if (i in x and i not in y)]
```

Python

- Python dilindeki kümeler bu iş için pratik kullanım sağlar. Kümeler dersindeki birleşim, kesişim, farkı işlemleri aşağıdaki gibidir.

```
x={"python", "java", "perl", "scala"}

y={"ruby", "python", "java","php"}


fark=x-y   # x kümesinde olup, y'de olmayanlar

birlesim= x | y # birleşim

kesisim= x & y #iki kümenin kesişimi

simetrik_fark = x^y #Simetrik fark= birleşim-kesişim
```

Python

- Python dilindeki kümeler bu iş için pratik kullanım sağlar. Kümeler dersindeki birleşim, kesişim, farkı işlemleri aşağıdaki gibidir. (***Not: kümeler çift değer içermezdi!***)

```
x={"python", "java", "perl", "scala"}

y={"ruby", "python", "java","php"}
```

```
: fark
{'perl', 'scala'}
:
: birlesim
{'scala', 'java', 'python', 'perl', 'ruby', 'php'}
:
: kesisim
{'java', 'python'}
:
: simetrik_fark
{'scala', 'perl', 'ruby', 'php'}
:
```

Zip: Liste örneği

- 9.yansıdaki örneği iki listeyi karşılıklı birleştirecek şekilde nasıl yapabiliriz?

```
x=[1,2,3]
y=["a","b","c"]

z=list(zip(x,y))
```

```
> z
[(1, 'a'), (2, 'b'), (3, 'c')]
>
```

Zip: Sözlük örneği

- Aynı örneği listeden biri key'ler, diğeri: value'ler olacak şekilde de birleştirebiliriz.

```
x=[1,2,3]
y=["a","b","c"]

z=dict(zip(y,x))
```

```
> z
{'a': 1, 'b': 2, 'c': 3}
```

Python Programlama Dili

Iterators, Generators, Counters

Yunus Santur

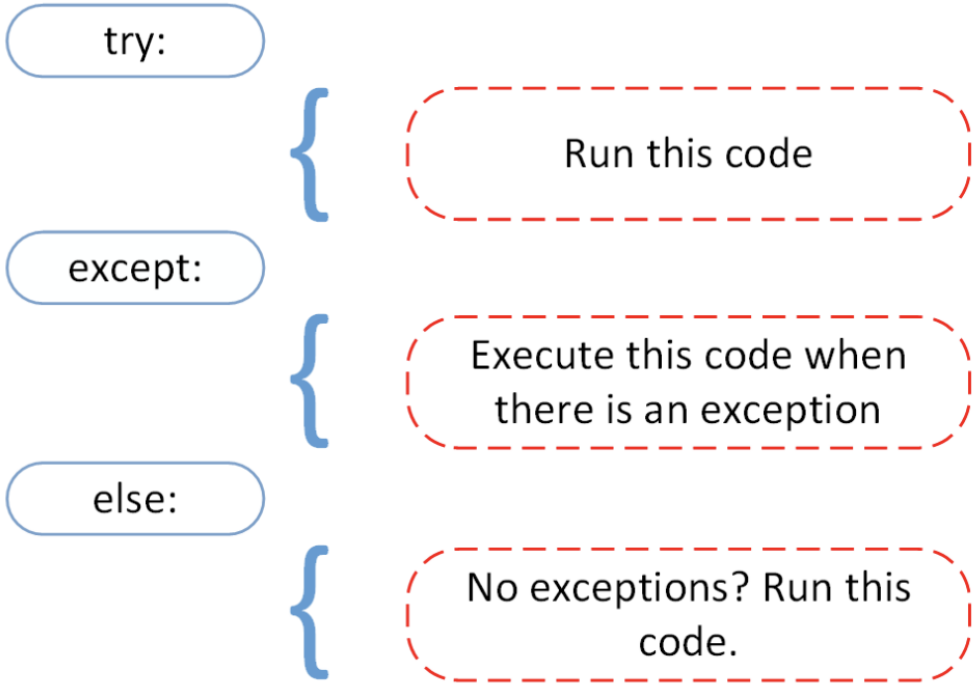
İstisna İşleme

- Gelecek örnekler için istisna işlemeye hızlı giriş yapıyoruz.
  - o'a bölme
  - Olmayan dosyaya yazma, okuma yapma
  - Veri tabanı işlemleri
  - Vb.

```
> print(5/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>
```

İstisna işleme mekanizması

- Genel yapı aşağıdaki gibidir.



İstisna işleme örnek

try: istisna oluşturması muhtemel kodlar.

except: istisna oluşması durumunda çalışacak kodlar (Programın hata verip yarıda kesilmesi engellenir.)

else: istisna oluşmadığında çalışacak kodlar.

```
try:
    a=5/0
except:
    print("istisna oluştu")
else:
    print(a)
```

Iterators

- Python dilinde iterators her nesneye iterasyon özelliği kazandırır.
  - list, set, tuple, dict zaten itere edilebilirlerdi.

```
x=[1,2,3]

for i in x:
    print(i)
```

Iterators

- next() her defasında bir sonraki öğeyi dönderir, öğe olmadığında exception (istisna) üretir.

```
y=iter(x)

next(y)
1
next(y)
2
next(y)
3
next(y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Iterators

- String üzerinde iterator

```
> a="Merhaba python"
> b=iter(a)
> next(b)
'M'
> next(b)
'e'
>
```

Iterators

- Neden kullanıyoruz
  - Avantaj
    - Listeler bir kerede tanımlanır ve bellek ayrılır, buna bağlı olarak diğer kaynaklar tüketilir, iter ile daha az kaynak tüketimi gerçekleştirilir. (veri tabanındaki tüm tablonun çekilmesi, diskteki tüm dosyaların okunması gibi)
  - Dezavantaj
    - Listenin sonuna, yada db'de ki son kayda erişmek için ***n*** sayıda iter çalıştırmak gerekir.

iter ve next

- Birbirleri yerine kullanılabilir.
- next() kullanımı daha pratiktir.

```
x=[1,2,3]
a=iter(x)
next(a)
1
a.__next__()
2
a.__next__()
3
```

Iter sınıf yazmak

- Aşağıdaki gibi yazılabilir.
- Sonsuz bir iter oluşturmuş oluyoruz.
- __ kullanımı sınıf metotlarını private yapar.

```
class Ciftler:
  def __iter__(self):
    self.a = 2
    return self

  def __next__(self):
    x = self.a
    self.a += 2
    return x

sinif = Ciftler()
x = iter(sinif)
```

Sonlu Iter sınıfı yazmak

- Bir önceki örnekten farkı sınıfın sonlu olmasıdır.
- Dersin başında istisna işlemeye bu yüzden değindik.

```
class Ciftler:
  def __iter__(self):
    self.a = 2
    return self

  def __next__(self):
    if self.a < 10:
      x = self.a
      self.a += 2
      return x
    else:
      raise StopIteration

sinif = Ciftler()
x = iter(sinif)
```

```
› next(x)
2
› next(x)
4
› next(x)
6
› next(x)
8
› next(x)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "main.py", line 12, in __next__
    raise StopIteration
StopIteration
› ^
```

Generators

- Iterator yaratmanın bir yoludur. Iter olabilecek bir nesneyi döndüren fonksiyondur.
- Normal metotlarda yer alan return yerine yield kullanmak iterator yaratmanın bir yoludur.
- return ifadesi metotu sonlandırırken, yield değeri döndürür, saklar ve fonksiyonu çağırmaya devam eder.

```
def test(a):
  a+=1
  yield a

  a+=1
  yield a

x=test(5)

print(next(x))
print(next(x))
print(next(x))
```

Generators vs List comprehension

- Liste üreticileri ile Generators arasındaki fark ***Lazy Evaluation*** dır.
  - Liste üreticiler için tek seferde gerekli kaynaklar tahsis edilir.
  - Generators tembel değerlendirme yapar, yani sırası geleni üretir, kaynakları dinamik kullanır.

```
> x=[i**2 for i in [1,2,3] ]
> type(x)
<class 'list'>
> x=(i**2 for i in [1,2,3] )
> type(x)
<class 'generator'>
> next(x)
1
> next(x)
4
> next(x)
9
```

Generators - örnek

- Bir önceki örneği metot ve yield kullanarak yapalım.

```
def kareAl(liste):
  for i in liste:
    yield i*i

x=[1,2,3,4,5]

y=kareAl(x)
```

```
> y
<generator object kareAl at 0x7fe1f855ceb0>
> next(y)
1
> next(y)
4
> next(y)
9
> next(y)
16
> next(y)
25
> next(y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Generators - örnek

- Aynı örneği liste değil, parametre üst sınırı olarak sabit düşünürsek aşağıdaki gibi kodlayabiliriz.

```
def kareAl(n):
  i=1
  while i<n:
    yield i*i
    i+=1
  else:
    raise StopIteration


y=kareAl(5)
```

Örnek1- fibonacci sayıları

- Fibonacci sayılarını iter() ile oluşturan NTP tabanlı bir kod.

```
class Fibo:
  def __iter__(self):
    self.f1=0
    self.f2=1
    return self

  def __next__(self):
    self.f1, self.f2 = self.f2, self.f1+self.f2
    return self.f2



fibos=Fibo()

x=iter(fibos)
```

Örnek2- fibonacci sayıları

- Fibonacci sayılarını yield kullanarak oluşturalım.

```
def fibGen(n):
  f1,f2=0,1
  i=0
  while i<n:
    f1,f2= f2, f1+f2
    yield f2
    i+=1
  else:
    raise StopIteration



x=fibGen(10)
```

Örnek3- fibonacci sayıları

- Liste üreteçleri ile fibonacci serisi oluşturalım.

```
> x=[0,1]
> y=[x.append(x[i-1]+x[i]) for i in range(1,10)]
> x
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
>
```

Lazy Evaluation

- Türkçeye tembel değerlendirme/yükleme olarak çeviriyoruz.
- Generators tembel yükleme yaparlar.
  - ✓Bellek etkinliği (sırası gelen üretilir, listelerde bellek ayrılır)
  - ✓Kaynaklar (sırası gelen için hesaplama yapılır, tüm liste için kaynak kullanılmaz)
  - ✓Uygulaması, kodlaması kolay ve zevklidir.
  - ✓Teorik olarak sonsuz veri akışı sunarlar.
  - ❑Her defasında çağrılmaları gerekir.

Diğer Kullanım alanları

- Veritabanı ve dosya işlemleri
- Sender – receiver
- Network, brodcast
- I/O işlemleri
- Sistem yönetimi-komut satırı
- Pipeline