İçindekiler

PYTHON101

Sayılar, Karakter Dizileri (Stringler) ve Değişkenler

-Sayılar kabaca ikiye ayrılır=int ve float

- int: tam sayı, kesikli.
- float: virgüllü sayı, sürekli.

-String: Karakter dizisi. Girilen ifadenin string olduğunu belirtmek için ' ' veya " " kullanılır.

-Not: Yazdığım kodların çıktısını ekrana yazdırmak istiyorsam print ifadesini kullanmak zorundayım.

örn: print("hello ai era")

-Elimdeki nesnenin tipini öğrenme:

type(9)--> int yazar.

type(9.2)--> float yazar

type("hello")--> str yazar.

Karakter Dizilerini Yakından Tanıyalım

-Kişisel notlar nasıl yazılır:

Yazının başına # konulur. Program bu yazıyı görmezden gelir.

-2 string ifadeyi toplarsam ne olur:

"a"+"b" yaparsam sonuç "ab" olur.

"a"*3 yaparsam sonuç "aaa" olur.

-STRING METODLARI

-len(): uzunluk bilgisi verir.

örn: gel_yaz="gelecegi_yazanlar"

len(gel_yaz)-->sonuç:17

-gel_yaz. yazdığımızda string tipindeki nesnemize uygulanabilecek metodlar karşımıza çıkar.
örn:capitalize, lower, replace vb.

-.upper(): bütün karakter dizisini büyük harfe çevirir.

örn: gel_yaz.upper()

-.lower(): bütün karakter dizisini küçük harfe çevirir.

örn: gel_yaz.lower()

-.islower(): bu ifade küçük mü diye sorar.

örn: gel_yaz.islower()

örn: gel_yaz.islower-->sonuç:true

örn: B=gel_yaz.upper()

gel_yaz.islower-->sonuç:false

-.isupper(): bu ifade büyük mü diye sorar.

örn: gel_yaz.isupper()

örn: B=gel_yaz.upper()

gel_yaz.isupper-->sonuç:true

NOT: Atanan değişkenler sağdaki variable explorer kısmında görünür. Hangi değişkene ne atadığımızı ve tipini görebiliriz.

-.capitalize():Baş harfi büyütür.

örn: gel_yaz.capitalize()--> sonuç: 'Gelecegi_yazanlar'

-.title(): Her kelimenin ilk harfi büyür.

örn: gel_yaz.title()--> sonuç: 'Gelecegi_Yazanlar'

-.replace(): Elimizdeki stringlerin içerisindeki karakterlerle ilgili değişiklik yapmak istediğimizde kullanılır.("neyi değiştirmek istiyoruz","ne ile değiştirmek istiyoruz") şeklinde yazılır.

örn:geleceği yazanlar içindeki e'leri a yapmak gibi.

örn: gel_yaz.replace("e","a")--> sonuç:"galacagi_yazanlar"

NOT: Atama işlemi yapmadığımız için orijinal değişkende değişiklik olmaz. gel_yaz değişkeni hala "gelecegi_yazanlar" şeklinde kalır. replace() gibi bazı metodlar orijinal değişkende değişiklik yapmaz. Bazıları kalıcı değişiklik yapabilir.

-.strip(): istenmeyen karakterleri kırpmak için kullanılır.

örn: gel_yaz=" gelecegi_yazanlar "

gel_yaz.strip()--> sonuç:"gelecegi_yazanlar" oldu etrafındaki boşluklar gitti.

NOT: .strip() metodu default olarak boşlukları kaldırmaya ön tanımlıdır. Kırpma işlemini boşluklara göre yapar. İstenirse bu ön tanımlı değer değiştirilebilir.

örn: gel_yaz="*gelecegi_yazanlar*"

gel_yaz.strip("*")--> sonuç: 'gelecegi_yazanlar' olur. Yıldızları kırp dedik.

-dir(): Elimizdeki veri tipine uygulanabilecek metodları gösterir.

örn: dir(gel_yaz)--> sonuç: birçok metod ismi çıkar.

-Sub String: elimizdeki string nesnenin alt birimlerine erişmek içim kullanır. Sadece string dğl ilerideki bazı veri yapıları içinde bu kullanılır. Köşeli parantez [] şeklindeki kullanılır. yani indexlerden yararlanılır. [0] gibi.

NOT: indexler 0'dan başlar.

örn: gel_yaz[0]--> sonuç:'g'

örn: gel_yaz[20]--> sonuç: indexerror: string index out of range hatası.

örn: gel_yaz[0:3]--> sonuç:'gel' 3'e kdr anlamına gelir. 0 dahil 3 dahil dğil.

örn: gel_yaz[3:7]--> sonuç:'ecegi' 7'ye kdr. 3 dahil 7 dahil dğil.

Değişkenler

-a=9 a bir değişken.Bu değişkenin tipi int.

-Değişken isimlendirme:

*python'da kullanılmayan kelimeler olmalı,

*yapılacak işlemi tanımlamalı.

-Değişkenler 2 tip:

*sayısal değişkenler: 3 tip(int,float,complex)

*string (karakter) değişkenler

type(100)--> sonuç:int

type(100.2)--> sonuç:float

type(1+2)--> sonuç:complex

Tip Dönüşümleri

-input(): kullanıcıdan bilgi almak için kullanılır.

örn:

toplama_bir=input()-->10

toplama_iki=input()-->20

toplama_bir+toplama_iki-->'1020'

veriyi kullanıcıdan str olarak alır.

örn:

int(toplama_bir)+int(toplama_iki)--> 30 int oldu.

örn: int(11.0) --> sonuç: 11

örn: float(12) --> sonuç: 12.0

örn: str(12) --> sonuç: '12'

type(str(12))--> sonuç: str

Kod Çıktısını Ekrana Yazdırmak: print Fonksiyonu

-print("hello")--> sonuç:'hello'

-print("gelecegi","yazanlar")--> sonuç:'gelecegi yazanlar'

-print("gelecegi","yazanlar", sep = "_")--> sonuç:'gelecegi_yazanlar'

-sep gibi şeylere"fonksiyonların genel amaçlarını biçimlendrmek için kullanılan alt görev belirticilere argüman denir". Yani sep print fonksiyonu altında bir argüman.

-print(girildiğinde otomatik olarak altında arguments gösterilir. Bu ayar Ayarlar-->help-->Editor tiklenerek açılır. Ayrıca sağdaki help kısmında da görülebilir. Ayrıca ?print ile de bakılır.

PYTHON201

Liste Oluşturma

#VERİ YAPILARI

#Listeler

1.Değiştirilebilir.

2.Kapsayıcı(farklı tipte verileri tutabilir).

3.Sıralıdır.

-Liste oluşturmak için 2 yol vardır.

1. []

2. list()

örn: notlar = [90,80,70,50]

type(notlar)--> sonuç: list

örn: liste = ["a",19.3,90]

örn: liste_genis= ["a",19.3,90,notlar]

liste içinde liste tuttuk.

len(list_genis)--> sonuç: 4

-liste içindeki elemanın tipini sorgulama:

örn: type(list_genis[0])-->sonuc:str

örn: type(list_genis[3])-->sonuc:list

-2listeyi birleştirip tek bir liste oluşturma:

örn:tum_liste=[liste,list_genis]

tum_liste-->sonuc: [["a",19.3,90], ["a",19.3,90,notlar]]

-liste silme:

del tum_liste

örn: liste=[10,20,30,40,50]

liste[1]-->snç: 20

liste[6]-->snç: indexerror

liste[0:2]-->snç:[10,20]

liste[:2]-->snç:[10,20]

liste[2:]-->snç:[30,40,50]

örn: yeni_liste=["a",10,[20,30,40,50]]

yeni_liste[2]-->snç:[20,30,40,50]

yeni_liste[0:2]-->snç: ["a",10]

yeni_liste[2][1]--> snç: 30

-liste elemanlarını değiştirme:

örn: liste=["ali","veli","berk","ayse"]

liste[1]="velinin_babasi"

liste-->snç: ["ali","velinin_babasi","berk","ayse"]

örn: liste=["ali","veli","berk","ayse"]

liste[0:3]="alinin_babasi","velinin_babasi","berkinbabasi"

liste--> snç: liste=["alinin_babasi","velinin_babasi","berkin_babasi","ayse"]

-listeye yeni eleman ekleme:

örn: liste=["ali","veli","berk","ayse"]

liste+["kemal"]

liste-->snç: ["ali","veli","berk","ayse"] ekleme işlemi kalıcı olmadı.kemal listeye eklendi ama kalıcı olarak eklenmedi.O yzden yeniden atama yapmamız lazım.

liste=liste+["kemal"]

liste-->snç: ["ali","veli","berk","ayse","kemal"]

-liste içindeki elemanı silme:

örn: del liste[2]

liste-->snç: ["ali","veli","ayse","kemal"]

-.insert(): indekse göre eleman eklemek içn kul.

örn: liste=["ali","veli","isik"]

liste.insert(0,"ayse")

liste-->snç: ["ayse","ali","veli","isik"]

-uzunluğunu bilmediğimiz bir listenin sonuna

eleman ekleme:

liste.insert(len(liste),"beren")

-.pop(): indekse göre listeden eleman siler.

örn: liste=["ayse","ali","veli","isik"]

liste.pop(0)

liste--> ["ali","veli","isik"]

-.count(): liste içindeki bir elemanın kaç tane old.nu

sayar.

örn: liste=["ali","veli","isik","ali","veli"]

liste.count("ali")--> snç:2

liste.count("isik")--> snç:1

-.copy(): listeyi kopyalar.

örn: liste_yedek=liste.copy()

-.extend(): 2 listeyi birleştirmek için kul.

örn: liste.extend(["a","b",10])

liste-->["ali","veli","isik","ali","veli","a","b",10]

NOT: .extend() ile listenin orijinal yapısında kalıcı

değişiklik meydana geldi.

-.index(): bir elemanın hangi indexte old.nu bulma:

örn: liste.index("ali")--> snç: 0

-.reverse(): elemanları terse çevirir:

liste.reverse()

liste--> snç: [10,"b","a","veli","ali","isik","veli","ali"]

-.sort(): sıralama yapar:

örn: liste= [10,40,5,90]

liste.sort()

liste--> snç: [5,10,40,90]

-.clear(): listenin içini temizler.

örn: liste.clear()

liste-->snç: []

Tuple Oluşturma

#Tuple'ların özellikleri:

1.Değiştirilemez.

2.Kapsayıcı (farklı veri tiplerini barındırır)

3.Sıralı

-tuple oluşturma:

1.()

örn: t=("ali","veli",1,2,3.4,[1,2,3,4])

2. parantezsiz yazma

örn: t= "ali","veli",1,2,3.4,[1,2,3,4]

3.tuple()

-tek elemanlı bir tuple oluşturma:

örn: t=("eleman")

type(t)-->snç:str

NOT: Tuplelara özgü bir özellik olarak, tek elemanlı tuplelarda tip tuple içindeki değerin tipini verir.

t=("eleman",) yaparsak bunun bir tuple oldunu anlar.

---

t=("ali","veli",1,2,3,[1,2,3,4])

t[1]-->snç: 'veli'

t[0:3]-->snç: ("ali","veli",1)

t[2]=99-->snç: typeerror:tuple değişikliğe izin vermez.

Sözlük Oluşturma

#Sözlüklerin özellikleri:

1.değiştirilebilir

2.kapsayıcı

3.sırasız

-anahtar ifadeler(key) ve anahtar ifadelerin(value) karşılıklarının bir arada tutulduğu referanslı veri yapısıdır.

-sözlük oluşturma:

1.{}

örn: sozluk={"REG": "Regresyon modeli",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"}

len(sozluk)-->snç: 3 yani eleman sayısı 3tür

2.dict()

örn: sozluk={"REG": 10,

"LOJ": 20,

"CART": 30}

örn: sozluk={"REG": ["RMSE",10],

"LOJ": ["MSE",20],

"CART": ["SSE",30]}

NOT: Sözlüklerde bir sıralama olmadığı için listelerdeki gibi bir index işlemleri yapılamaz.

örn: sozluk[0]-->snç: keyerror

örn:sozluk["REG"]-->snç:'regresyon modeli'

örn: sozluk={"REG": {"RMSE": 10,

"MSE": 20,

"SSE": 30},

"LOJ": {"RMSE": 10,

"MSE": 20,

"SSE": 30}}

sozluk["REG"]["SSE"]-->snç: 30

-sözlüğe eleman ekleme:

örn: sozluk={"REG": "Regresyon modeli",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"}

sozluk["GBM"]= "Gradi Boos Mac"

sozluk--> snç: {"REG": "Regresyon modeli",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"

"GBM": "Gradi Boos Mac"}

-sözlükteki elemanı değiştirme:

örn: sozluk["REG"]= "coklu regresyon"

sozluk-->snç: {"REG": "coklu regresyon",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"

"GBM": "Gradi Boos Mac"}

örn: sozluk[1]="yapay ag"

sozluk-->snç: {"REG": "coklu regresyon",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"

"GBM": "Gradi Boos Mac"

1: "yapay ag"}

NOT: sözlüklerde key değerleri ancak sabit veri yapıları ile oluşturulur. (int,str, tuple gibi)

örn: l=[1]

sozluk[l]="yeni bir sey"--> snç: typeerror:unhashable

örn: t=("tuple",)

sozluk[t]="yeni bir sey"

sozluk--> snç: {"REG": "coklu regresyon",

"LOJ": "Lojistik regresyon",

"CART": "Classificaiton"

"GBM": "Gradi Boos Mac"

1: "yapay ag"

("tuple",): "yeni bir sey"}

Set Oluşturma

#setler(kümeler):

1.değiştirilebilir

2.kapsayıcı

3.sırasız

4.değerleri eşsiz (tekrar eden değerlerden oluşamaz.)

NOT: sırasız olduğu için index işlemleri yapılamaz.

-set oluşturma:

1.set()

örn: s=set()

örn: l=[1,"a","ali",123]

s=set(l)

s--> snç: {1,"a","ali",123}

örn: t=("a","ali")

s=set(t)

s-->snç: {"a","ali"}

-setler her karakterden yalnız 1 tane barındırır:

örn: l=["ali","lutfen","git","git","ali","git"]

s=set(l)

s-->snç: {"ali","lutfen","git"}

len(s)-->snç: 3

örn: l=["gelecegi","yazanlar"]

s=set(l)

s-->snç: {"gelecegi","yazanlar"}

-.add(): sete eleman ekler:

örn: s.add("ile")

s--> snç: {"gelecegi","ile","yazanlar"}

NOT:elemanlar sırasız biçimde setlerde saklanır.

setlerde metodlara ulaşma:

-dir(s)

-.remove(): setten eleman siler.

s.remove("ile")

s--> snç: {"gelecegi","yazanlar"}

-.discard(): set içinde bulunmayan elemanı silmek istersek hata verir.keyerror verir. Bu hata çıkmasın istersek remove yerine discard kullanılır.

örn: s.discard("ile")--> içinde bulamadığı elemanı silmeaz ama hata da vermez

---

-difference(): 2 kümenin farkı yada "-"

-intersection(): 2 küme kesişimi yada "&"

-union(): 2 küme birleşimi.

-symmetric_difference(): 2sinde de olmayanlar.

-.difference(): fark

örn: set1=set([1,3,5])

set2=set([1,2,3])

set1.difference(set2)--> snç: {5}--> set1 den olup set2 de olmayan değerler gelir.

set2.difference(set1)--> snç: {2}

örn: set1-set2--> snç: {5} fark bu şekilde de alınabilir.

örn: set2-set1--> snç: {2}

-symmetric_difference(): 2sinde de olmayan

örn: set1.symmetric_difference(set2) --> snç: {2,5}--> 2sinde de olmayan gelir.

-.intersection(): kesişim.

örn: set1.intersection(set2) --> snç: {1,3}

örn: set1&set2 --> snç: {1,3} bu şekilde de alınır.

-.intersection_update(): kesişim elemanlarını başka bir değişkende tutmak istemiyorum. direk set1 i değiştirmek istiyorum. o zmn kullanılır.

örn: set1.intersection_update(set2)

set1 --> snç: {1,3}

-.union(): birleşim.

örn: set1.union(set2)--> snç: {1,2,3,5} --> set old. için her elemandan 1 tane tutar.

#setlerde sorgu işlemleri:

set1=set([7,8,9])

set2=set([5,6,7,8,9,10])

-.isdisjoint(): 2kümenin kesişiminin boş olup olmadığını sorgulama:

örn: set1.isdisjoint(set2)--> 2 kümenin kesişimi boş mu?diye sorar.--> snç: false--> boş dğl.

-.issubset(): bir kümenin bütün elemanlarının başka bir küme içerisinde yer alıp almadığını sorgulama:

örn: set1.issubset(set2)-->set1 set2nin alt kümesi midir? diye sorar.--> snç: true

-.issuperset(): bir kümenin diğer kümeyi kapsayıp kapsamadığını sorgulama:

örn: set2.issuperset(set1)-->set2 set1 kapsiyor mu? diye sorar.--> snç: true

---

PYTHON301

Fonksiyonlar

#fonksiyon dökümantasyonuna ulaşma:

?print-->print fonk.nun dökümantasyonunu getirir. Kullanımını, argümanlarını vb. gösterir.

#Matematiksel işlemler:

-4*4

-4/4

-5-1

-6+3

-3**2: kuvvet alma

-fonksiyon tanımlama:

```
def fonk. ismi (kullanılacak argümanlar):
    fonksiyonun yapacağı işler
```

örn:
```
def kare_al(x):
        print(x**2)

    kare_al(3)
```

NOT: fonksiyon sonucunu ekrana yazdırmak yerine çıkan değer üzerinde işlem yapmak istersek return kullanılır.

#bilgi notuyla çıktı alma:

örn:
```
def kare_al(x):
        print("girilen sayinin karesi:" + str(x**2))

    kare_al(3)
```

örn: def kare_al(x):

```
    print("girilen sayi:" + str(x)+ " karesi:"+ str(x**2))
```

kare_al(3)

#2 argümanlı fonk. tanımlama:

örn: def carpma_yap(x, y):

```
    print(x*y)
```

carpma_yap(2,3)--> snç:6

#ön tanımlı argümanlar:

-sep: ön tanımlı ol. boşluk bırakır.

örn: def carpma_yap(x, y = 1):

```
    print(x*y)
```

carpma_yap(2)--> snç:2(y ön tanımlı olmuş oldu.)

#argümanların sıralaması:

örn: def carpma_yap(x, y = 1):

```
    print(x*y)
```

carpma_yap(y = 2,x = 3)--> argümanların sırasını karıştırdıysak bu şekilde belirtebiliriz.

#Ndn fonk kullanılır:

-tekrar eden görevleri yerine getirmek

-varolan işleri daha kolay gerçekleştirmek

#Fonksiyon Çıktılarını Girdi Olarak Kullanmak: return

örn: def direk_hesap(isi, nem, sarj):

print((isi + nem)/ sarj)

cikti = direk_hesap(25, 40, 70)

cikti---> snç: cikti--> fonk çıktısı direk kullanılamaz!

örn: def direk_hesap(isi, nem, sarj):

return ((isi + nem)/ sarj)

cikti = direk_hesap(25, 40, 70)

cikti---> snç: 0,928--> fonk çıktısını kullanmak için return kullanılır.

NOT: fonk.lar return ifadesine geldiğinde durur. Aşağısındaki işlemlerle ilgilenmez. kullanırken buna dikkat!

#Local ve Global Değişkenler:

örn:

x= 10 --> bu x ve y global.

y= 20

def carpma_yap(x, y):--> buradaki x ve y

local

değişkendir. yani

fonksiyonun

veya döngünün

etki

alanındadır.

return x*y

carpma_yap(2, 3)

#Local Etki Alanından Global Etki Alanını Değiştirmek:

örn:

x= [] --> boş bir liste, global etki alanında.

```
def eleman_ekle(y):
    x.append(y)-->listeye eleman ekle.
    print(str(y) + "ifadesi eklendi")

eleman_ekle("ali")-->ali listeye eklendi.
```

NOT: python öncelikli olarak local etki alanındaki değişkenleri tarar, arar ve bulmaya çalışır. yani x i önce local alanda arar, localde bulamazsa global alanı tarar.

Karar Kontrol Yapıları

#karar kontrol yapıları/koşul yapıları:

örn: sinir= 500

sinir== 400--> sınır 400 mü? diye sorar.--> snç: false

örn: 5==4--> false

#İF-ELİF-ELSE yapısı:

örn:

```
sinir = 50000
gelir = 40000

if gelir == sinir :
    print("gelir sinira eşit")
elif gelir < sinir :
   print("gelir sinirdan küçük")
```

```
else :
    print("gelir sinirdan büyük")
```

#Uygulama: if ve input ile Kullanıcı Etkileşimli Program:

mağazanın ismi ve gelirini kullanıcıdan al. bir karar ver.

örn:

```
sinir = 50000
magaza_adi = input("magaza adı gir: ")
gelir = int(input("gelir gir: "))

if gelir>sinir:
    print("tebrikler:" + magaza_adi + "promosyon aldınız")
elif gelir<sinir:
    print("uyarı:" + str(gelir) + "düsük")
else:
    print("sinirdasin")
```

NOT: kullanıcıdan gelen bilgi her zmn str olarak gelir. Bunu int olarak işlemek istersek dönüştürmek zorundayız.

Döngüler

#DÖNGÜLER:

#FOR döngüsü:

örn:

```
ogrenci = ["ali", "veli", "isik", "berk"]
ogrenci[0]
```

for i in ogrenci : -->i liste içindeki her bir

elemanı gezmek ve temsil
etmek için kullandığımız
geçici bir değişken.
ogrenci listesi içindeki
her bir eleman için dön

print(i)--> snç: ali
veli
isik
berk

örn:

maaslar = [100,200,300,400,500]

for maas in maaslar:
    print(maas)--> snç: 100
200
300
400
500

#Döngü ve Fonksiyonların Birlikte Kullanımı:

örn: patron maaşlara %20 zam yapmak istiyor.

maaslar = [100,200,300,400,500]

def yeni_maas(x):
    print(x*20/100 + x)

yeni_maas(100)

for i in maaslar:

```
    yeni_maas(i)
```

#Uygulama: if, for ve Fonksiyonların Birlikte Kullanımı:

örn:patron maaşlara %20 zam yapmak istiyor. Ama maaşı 300tl veya yüksek olanlara %10, 300tlden az olanlara %20 zam yapılacak.

```
maaslar = [100,200,300,400,500]

def maas_ust(x):
    print(x*10/100 + x)

def maas_alt(x):
    print(x*20/100 + x)

for i in maaslar:
    if i >= 300:
        maas_ust(i)
    else:
        maas_alt(i)
```

#Break ve Continue:

-break: belirli bir şart sağlandığında döngü bitirilir

-continue: şartı sağlayan eleman görmezden gelinir.

örn: maası 300tlye kdr olanlarla ilgilenmek istiyorum. maas 300e gelince döngü dursun.

```
maaslar = [800,500,200,100,300,700,100]

maaslar.sort()

for i in maaslar:
```

```
    if i ==300:
        print("kesildi")
        break-->döngü dursun.
    print(i)-->300e gelene kdr for döngüsünün yakaladığı elemanları görmek için.
-->snç: 100
          100
          200
          kesildi
```

örn: maas 300e gelince bununla ilg bir işlem yapmayalım.

```
maaslar = [800,500,200,100,300,700,100]

maaslar.sort()

for i in maaslar:
    if i ==300:
        print("kesildi")
        continue-->döngü görmezden gelsin
    print(i)
-->snç: 100
          100
          200
          500
          700
          800-->300 değeri atlandı.
```

#WHILE:

-şart sağlandığı sürece demektir.

örn:

sayi = 1

while sayi< 10: --> sayi 10dan < old sürece.

    sayi += 1   --> sayi = sayi +1 demek

    print(sayi)

--> snç: 2-3-4-5....9-10

PYTHON401

Nesne Yönelimli Programlama

#Nesne Yönelimli Programlama (OOP)

#Sınıflara Giriş ve Sınıf Tanımlamak:

-Sınıf: benzer özellikler, ortak amaçlar taşıyan, içerisinde metod ve değişkenler olan yapılar.

-class tanımlama:

örn:

```
class VeriBilimci():
    print("bu bir sınıf")
```

#sınıf özellikleri(class attributes):

örn:

```
class VeriBilimci():
    bolum = ' '
    sql = ''
    deneyim = 0
    diller = []
```

```
VeriBilimci.bolum
VeriBilimci.sql-->snç: evt
```

-sınıfların özelliklerine erişmek:

örn: VeriBilimci.bolum

-sınıfların özelliklerini değiştirmek:

örn: VeriBilimci.sql = "hayir"

VeriBilimci.sql--> snç: hyr olarak değşti

#Sınıf Örneklemesi (instantiation):

ali = VeriBilimci() -->ali bu sınıfın bir örneği.yani bu sınıfın özelliklerini taşıyor.

örn:

ali = VeriBilimci()

ali.sql--> snç:hyr

ali.deneyim--> snç:0

ali.diller.append("python")

ali.diller-->python

veli = VeriBilimci()

veli.sql--> snç:hyr

veli.diller-->python-->1 kişide yapılan değişiklik tüm sınıfa oldu.

#Örnek Özellikleri:

örn:

```
class VeriBilimci():
    diller = ["R", "Python"]-->sınıf özelliği
    bolum = ' '
    sql = ''
    deneyim = 0
    def __init__(self):
        self.diller = [] --> Herbir örneğin kndi içinde değişen özelliklerden oluşabildiği bilgisini verdik. Örnek özelliği tanımladık.
        self.bölum = ' '
```

ali = VeriBilimci()

ali.diller-->snç: []-->boş

veli= VeriBilimci()

veli.diller-->snç: []-->boş

ali.diller.append("python")

veli.diller-->snç: []-->boş-->etkilenmedi.

veli.diller.append("R")

veli.diller-->snç: ['R']

VeriBilimci.diller-->snç: ["R", "Python"]

ali.bolum = "istatistik"

VeriBilimci.bolum-->snç:boş

veli.bolum-->snç:boş

veli.bolum = "EE"

veli.bolum-->snç: EE

ali.bolum-->snç:istatistik

VeriBilimci.bolum-->snç:boş

-def __init__(self):-->self örnekleri temsil eder. Örnekleyerek oluşturduğumuz ali ve veli sınıf örneklerinin temsil edilmesini, onlara işlem yapılmasını sağlayan temsilcidir.

#Örnek Metodları:

örn:

```
class VeriBilimci():
    calisanlar = []
    def __init__(self):
        self.diller = []
        self.bolum = ' '
    def dil_ekle(self, yeni_dil):-->fonksiyon
                                    çalışsın ama
                                     örneklerin
```

özelinde

çalışsın.

```
        self.diller.append(yeni_dil)

ali = VeriBilimci()
ali.diller--> []
ali.bolum--> ' '

veli= VeriBilimci()
veli.diller--> []
veli.bolum--> ' '

ali.dil_ekle("R")
ali.diller--> ['R']

veli.dil_ekle("python")
veli.diller--> ['python']
```

#Miras Yapıları(inheritance):

örn:

```
class Employees():
def __init__(self):
        self.FirstName = ""
        self.LastName= ""
        self.Address= ""

class DataScience(Employees):
def __init__(self):
        self.Programming= ""
```

```
class Marketing(Employees):
def __init__(self):
        self.StoryTelling= ""

veribilimci1 = DataScience()
veribilimci1.FirstName-->bu özellikler miras geldi.

mar1 = Marketing()
mar1.LastName-->bu özellikler miras geldi.
```

-Önceki örnek sabit değerlerden oluşuyor. Sıradaki örnek ise daha fonksiyonel. Bir sınıfı tıplkı bir fonksiyon gibi belirli argümanlar alacak şekilde tanımlama :

```
class Employee_yeni():
    def __init__(self, FirstName, LastName, Address ):
        self.FirstName = FirstName
        self.LastName= LastName
        self.Address= Address

ali = Employee_yeni("a", "b", "c")
ali.Address-->'c'
```

Fonksiyonel Programlama

#Fonksiyonel Programlama:

-nesne yönelimli programlamada her şey nessnelerden oluşur. Fonksiyonlar, stringler vs. her şey nesne idi.

-fonk programlama da ise fonksiyonlar daha ön planda.

-fonk.lar dilin baştacıdır.

-fonkların daha esnek çalışma yapısı vardır.

-1.sınıf nesnelerdir.

-yan etkisiz fonksiyonlar(stateless, girdi-cikti)

-yüksek seviye fonk.lar--> dha az çaba ile daha çok iş yopmayı sağlar.

-vektörel operasyonlar

#yan etkisiz fonk.lar(pure functions):

örn: Bağımsızlık/yan etki

```
A=5

def impure_sum(b):
    return b + A

def pure_sum(a, b):
    return a + b
```

impure_sum(6)--> 11-->A=9 yaparsak snç:15 olcak. Yani fonk. sounucu başka etkene bağlı olarak değişiyor.

pure_sum(3,4)-->7--> A yıda değiştirsek bu fnk.nun sonucu değişmez. Sonucun her zmn 7 olacağından eminiz. Ancak bir girdi verdiğimde bir çıktı üreteceğini ve bunun aynı olacağı biliyoruz.

-OOPdaki pure olmayan(saf olmayan) fonk ve ölümcül yan etkili fonk.lar sınıfların içinde tanımlanan yapıların birbirlerini etkilemesi anlamına gelir.FPda ise bağımsız birbiri içinde,peipeşe kullanılanbililen,ancak bir girdi verildiğinde çıktı üreteceğini bildiğimiz fonksiyonel yapı.

#isimsiz fonksiyonlar:

-isimli olan örn:

```
def old_sum(a, b):
    return a+b

old_sum(4, 5)
```

örn: isimli

```
nem_sum = lambda a,b: a+b
nem_sum(4, 5)
```

örn:isimsiz

```
sirasiz_liste = [('b',3),('a',8),('d',12),('c', 1)]
sirasiz_liste
```

sorted(sirasiz_liste , key = lambda x: x[1]) --> lambda fonkiyonu tanımladım ama bir değişkene atamadım isimlendirme yapmadım.--> çalıştırınca--> snç: sıranmış liste

---

#Vektörel Operasyonlar:

#OOP

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

ab = [ab]

for i in range(0, len())--> 0dan anın içinde ne kdr eleman varsa o kdr gezin.
    ab.append(a[i]*b[i])
```

#FP

```
import numpy as np
a = np.array([1,2,3,4])
b = np.array([2,3,4,5])
```

a*b-->snç: [2,6,12,20]

#Map, filter ve reduce Fonksiyonları

örn:

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

for i in liste:

    print(i+ 10)

#map(): verilen vektörün içinde belirli bir fonk.nu çalıştırma, işlem yapma mkanı verir.

örn:

list(map(lambda x: x+10, liste)) --> her bir elemanın üzerine 10 değeri eklensin, nerede yapsın liste de yapsın.--> snç: [11,12,13,14,15]

#filter(): fonk ve iteratif bir nesne alarak çalışır. aranan şartı sağlayan tüm elemanlar filtrelenir. Dğerlerle bir işlem yapmaz, sadece şartı arar filtreler.

örn:

liste = [1,2,3,4,5,6,7,8,9,10]

list(filter(lambda x: x %2 == 0, liste))--> snç: [2,4,6,8,10]

#reduce(): map ve filter a benzerdir fakat indirgeme işlemi yapar.

örn:

from functools import reduce

liste= [1,2,3,4]

reduce(lambda a,b: a + b, liste)--> snç:55

Modül Oluşturmak ve İstisnalar

#Modül Oluşturmak ve İstisnalar:

#Modül/kütüphane/paket Oluşturmak:

-başka bir sfdan yeni_maas fonk.nu çağıracağız:

HesapMudulu.py dosyası içinde:

def yeni_maas(x):

```
    print(x*20/100 + x)

maaslar = [1000,2000,3000,5000]
```

örn1: başka bir çalışma dosyasına yazıyoruz:

```
import HesapModulu

HesapModulu.yeni_maas(1000)--> snç:1200
```

örn2:

```
import HesapModulu as hm
hm.yeni_maas(1000)--> snç: 1200
hm.maaslar --> snç: [1000,2000,3000,5000]
```

örn3:

```
from HesapModulu import yeni_maas
yeni_maas(1000)--> snç: 1200
```

#Hatalar (İstisnalar) :

1.programcı hatası: basit hatalar

2.program hatası (bug): zor, kritik hatalar

3.istisna hatalar(exceptions): çalışmaya dvm et demenin yoludur.

örn:ZeroDivisionError hatası

```
a=10
b=0

a/b--> snç:ZeroDivisionError

try:
    print(a/b)
```

except ZeroDivisionError:-->hatanın tipini belirttik. bilgi notu yaz ama programın akışını bozma

    print("payda da 0 olmaz")

örn: tip hatası

a=10

b="2"

a/b--> TypeError

try:

    print(a/b)

except TypeError:

    print("sayi ve string hatası")

Bölüm Sonu Değerlendirmesi

#Neler öğrendik:

-python giriş

-çalışma ortamı ayarları

-temel hareketler

-veri yapıları

-fonk.lar ve fonk. okur yazarlığı

-kontrol & karar yapıları

-döngüler

-nesne yönelimli programlama(OOP)

-fonksiyonel programlama (FP)

-modüller

-hata yönetimi