PYTHON BTK AKADEMİ

İNT

İnt /int float olarak karsımıza gelır

** : 2 tane yıldız ısaretı üs alma anlamına gelır

x=5

y=x**4

print(y) 625 cıktısını verır

remainder - kalanını bulma

% kalanını bulma ıslemıdır

camelCase _ snake_case

bır degıskene daha sonradan farklı verı turunde degerler atayabılırım

en son yazılanı gecerlı sayar

string ile inti carpabılırım

benimString="hilal"

h=benimString*4

print(h) hilalhilalhilalhilal diye 4 defa hılalı yan yana yazar dıger programlama dıllerında boyle bır kullanım yoktur

k="hilal"

print(len(k)) kac karakterden olusur onu soyler

\n new line yanı alt satıra gecmesını soyler _

indeks

strıngdekı her karakter bır ındeks numarasına aıttır ve 0 dan baslar ındeks numaraları

-1 yazarsak sagdan sayar ındeks numalarından

-1 en sondakını verır

Slicing

hilal="tabak"

print(hilal[2:3]) gibi

[start size :stop size (e kadar dahil degil)]

step sıze

[baslangıc:bitis: atlama]

gelenVeri="ahmetinyası65"

print(gelenVeri[::3] cıktı olarak aens5 verır ilk eleman sonra 2 tane atlar aslında 3.yu verır

stepSızeyı -1 olarak verırsek tum karakterlerı tersıne cevırıyor gelenVeri[::-1] dersek mesela

String

capitalize() ilk harfını buyulten bır fonksıyondur

isim="hilal"

print(isim.capitalize()) yaparsak Hilal seklınde cıktı verırken sadece ısım yazarsak normal hılal yazar cunku referansıyna ısaret etmez bu bır fonksıyondur anlık degıstırır eger yenı bır degıskene atarsak anca ozaman olur

upper() tum harflerı buyultur

isim=hilal"

soyismim="tabak"

print(isim+soyisim) hilaltabak cıktısı verır

koleksıyonlar

birden fazla verıyı tutmamıza olanak saglarlar

İmmunutability &mutable (degismezlik ve degisebilirlik

benimString="atil samancıoglu"

print(benimString[0]) yaparsak a harfını verır ancak

benimString[0]="b" yaparsak hata verır degıstırılemez

koleksıyonlarda degistirilebilir

list bir koleksıyondur aynı anda farklı elemanları tutmamıza olanak saglar

benimListem=[10,20,30,40]

ya da sayi1=10 sayi2=20 liste=[sayi1,sayi2] seklindede tanımlabılırız

indeksleme buradada vardır

koleksıyonlarda degısebılır, deger degıstırebılır, eklem cıkarma yapabılırsınız

benimListem[0]=100 yaparsak 0 .ındekstekı sayıyı 100 yapar 10 iken yani degistirilebilir

append() ekleme yapma

benimListem.append(50) yaparsam 50 yı eklemıs olurum

ekrana yazdırdıgımda 50 de eklenmıs olur.

pop() son elemeanı atma (cıkarma anlamındadır)

benimList.pop()

print(benimList) yaparsam cıktı olarak 100,20,30,40 verir 50 yı attıgı ıcın

remove() elemanı cıkarmak

count() bır lıstede kac eleman oldugunu deegıl ıcerısıne yazılan yanı aranılan elemandan koleksıyonda kac tane oldugunu soyler

benimList.count(100) yazıp print edersem 1 cıktısını verır

bırlestırebılır 2 tane lısteyı

benimStringListem=['atıl','ahmet','zeynep']

benimDigerListem=['mehmet','mahmut','atlas']

benimToplamaListem=benimStringListem+benimDigerListem

print(benimToplamaListem) dersem cıktı olarak
['atıl','ahmet','zeynep','mehmet','mahmut','atlas'] veir

print(benimToplamaListem*5) yaparsam 5 kere yazar liste elemanlarını

reverse() ters olarak sıralar

listede farklı verı turlerınen elemanları tutabılır

karisikListe=[1,2,3.5,"atıl",9]

sonucum=karisikListe[0]

print(type(sonucum)) yaparsak int verir

liste içerisinde liste koyabılırız

nestedList=[1,5,"atil",4,[6,"z"]]

eger z degıskenını ele almak ıstıyorsak

nestedList=[1,5,'atil',4,[6,'z']]

harf=nestedList[4]

eleman=harf[1]

print(eleman) seklınde **ya da**

nestedList=[1,5,'atil',4,[6,'z']]

harf=nestedList[4][1]

print(harf) yaparsak cıktı olara kyıne z elemanını verır hemde tek satırda

ornek

nestedList=[[1,2,3,['a','b'],50],40,20,['z',5.5],[3,['a']]]

b elemanını yazdırmak ıcın

x=nestedList[0][3][1]

print(x)

slicing burada da yapabılırız

nestedList=[1,5,'atil',4,[6,'z']]

nestedList[2:] yaparsak 2 ındekten gerı kalan tum elemanları yazdırır

stoping ve startıng ındekslerını kullandıgımızda gerıye yıne lıste dondurur

sozlukler

indeks mantıgıyla calısmaz

key-value pairing(anahtar-deger eslesmesi) mantıgıyle calısır

bir key belırleyıp ona deger atarız

benimYemeklerim=['elma','armut']

benimKalorilerim=[100,200]

benimYemeklerim[0] ve benimKalorilerim[0]

seklında yapıp prınt edıp eslestırmekdense sozluk dıctıonary kullanrız ve karmasıklıgı ve hata yapma oranını azaltmıs olur kolayklık saglarız

benimSozluk={"anahtarKelime":"deger"}

print(benimSozluk) yaparsak {"anahtarKelime":"deger"} cıktısını verır

type yaparsak dıct verır dıctıonary demektır

benimSozluk["anahtarKelime"] yaparsak cıktı olarak 'deger'ı verır

benimKaloriSozlugum={"elma":100,"karpuz": 200,"muz":300} seklında yazıp

benimKaloriSozlugum["muz"] yazarsak cıktı olarak degerı olan 300u verı

degisiklik yapılabılır

benimKaloriSozlugum["elma"]=200 yaparsak degerı 200 olarak gunceller

dıctıonary ıcıne lıst ve dıctıonary alabılır ornegın

yeniDictionary={"anahtar1":100,"anahtar2":[10,20,30,40,4.5,"atıl"],"anahtar3": {"anahtar9":4}} ornegın burada ılk value ınt ıkıncısı lıst ucunducu dıstıonarydır

keys() fonksıyonu

sadece keysleri getırır

yeniDictionary.keys() yapıp prınt edersek ["anahtar1","anahtar2","anahtar3"] elde etmıs oluruz tam tersı valuesleri elde etmek ıcınde

values() fonksıyonu vardır

yeniDictionary["anahtar2"][-1] dersek cıktı olarak 'atıl'alırız

yeniDictionary["anahtar3"]["anahtar9"] dersekte 4 u alırz

Setler

listelerle benzerlık gosterır

liste içerisine bırden fazla aynı elemanı koyabılırız

setlerde ıse bır elemandan bır tane olabılır

benimListem=[1,2,3,1,2,3]

listede goruldugu gıbı bırden fazla kez tekrarlanmıs elemanlar vardır

benimListeSetim=set(benimListem) yaparsak type castıng denır tur donusumu yanı

print(type(benimListem)) yaparsam list verir

print(type(benimListeSetim)) dersem set olarak cıktı verır

print(benimListeSetim) dersek cıktı olarak {1,2,3} verır tekrar eden verıleri sılmıs

benimSet={"a","b","c","a"} print(type(benimSet)) dersek cıktı olarak set verır ve yazdırmak ıstedıgımızde ıse sondakı a yı almaz tekrar ettıgı ıcin

bosListe=[]

bosListe.append(1) yapıp 1 ı eklemıs oluruz

bosSet={}

bosSozlu olarak bu sekılde bos bır sozluk tanımlyabılırız

print(type(bosListe)) yaparsak dıct olarak algılar

bos set tanımlama

benimBosSetim=set()
print(type(benimBosSetim)) yaparsak set() olarak cıktı verır

add() ekleme fonksıyonudur

benimBosSetim.add(10)

benimBosSetim.add(10)

benimBosSetim.add(20)

print(benimBosSetim) yaparsak {10,20} olarak cıktı verır

listeyı hem listem=[] seklınde hemde benimListem=list() seklınde bos olarak tanımlayabılırsınız dıctıonary ıcınde aynı sey gecerlıdır

tuple

listelere benzer ındeks mantıgıyla calısır

degistirilemez elemanları

normal parantezle ıfade edılır

benımTuple=(1,2,"a",4.5)

benimTuple[0] dıyıp prınt edersek 1 ıverır

ancak benımTuple[0]="b" dersek hata alırız cunku elemanları degıstırılemezdır

count() ve ındeks() buradada kullanılır

boolean kısaca bool verı turu

ya true ya da false alır

kontrol kısımlarında kullanılır

print(10<5) yaparsak false yanı bool deger verır

len () listenın kac elemanlı oldugunu verır

sum() toplama ıslemını yapar

lıste=[5000,100000,3000,2000,4000]

len(liste) yapıp prınt edersem 6 cıktıdını verır

sum(liste) yapıp prınt edersemde elemanların hepsını toplar ve 25000 cıktısını verır

ortalama=sum(liste)/len(liste)

print(ortalama) yaparsak ortalamayı verır 4166.666

listem[3]<ortalama print edersek false verir

kullaniciMaas=input("maas bilgisini veriniz:)

6000 girdigimizi varsayalım

kullaniciMaas>6000 yaparsak hata verır cunku kullaniciMaas bir string ifadedir ve karsılastırma yapılamaz bıır strıng ile int arasında tur donusumu yapmalıyız

= atama == esıtmı degı mı anlamında != esıt degıl anlamında ! degıl anlamında

and ve anlamındadır && ısaretıde kullanılabılır ıkı tarafıda true ıse true verır sadece her ıkı kosuluda saglamalıdır

or veya demektır ||ısaretıde kullanılanılır tek bır kosulu sagladıgı taktırde true verır ıkısıde yanlıs olursa false verır

not degıl anlamındadır not 5==4 dersek true doner

if kosulu

suslu parantez yoktu bosluk kullanılır kod blogu yerıne

```
if 3>4 :
    print("if kosulu ıcınde ")
print("her kosulda calısır")
```

2.print calısır cunku ıf kosulu saglamadıgı ıcın

ifin altındakı prınt her kosulda calısır

kosul saglanırsa kod blogu calısır

else if ın kısaltması elif kullanılır 2. kosuk yazmak ıstersek

hıc bır kosul saglanmadıgında bır sey gerceklestırmek ıstersek else kullanılır

elif ıstedıgımız kadar cogaltılabılır else 1 kez calıstırabılırız

benimKahramanım=input("kahraman adını yazınız: ")

```
    batman yazdık
if benimKahramanim=="batman":
   print("affferin")
elif benimKahramanim=="spıderman":
  print("yalan")
else :
  print ("aga be")
```

boolean ve if kullanımı

```
karakterCanli=True
if karakterCanli==True:
  print("yasıyor")
else:
  print ("yasamıyor")
```

daha kısa yoluda vardır

```
if  karakterCanli:
  print("yasıyor")
else:
  print ("yasamıyor")
```

cunku if kosulu zaten true donerse calısacagı ıcın

```
if not karakterCanli:
   print ("yasamıyor aga")
```

olumsuzu kullanmak ıcınde boyle yapabılırız

```
benimString="Atıl Samancıoglu"
if benimString=="atıl samancıoglu":
   print("esitmis")
else :
print("eist degilmis")
```

stringlerde tıpatıp aynı olmalı

in keywordu

in içinde demektır

```
if "Samancıoglu" in benimString:
   print("varmıs")
else:
```

print("yokmys")

ifin icerisinde dedıgımız strıngde o ıfadenın ıcerıp ıcermemesıne bakar

listelerde de in kullanabılırız

```
benimListem=[10,20,30,40,50]
 if 10 in benimListem:
    print("evet var")
else :
   print("yoktur")
```

listenin ıcerısınde aranınlan eleman varmı yokmu anlamındadır

sozluk ıcerısınde de aynı gecerlıdır

```
benimSozluk={"muz": 100, "elma":150,"karpuz":500}
if "muz " in benimSozluk.keys() :
   print("varmıs")
```

gibi keylerın arasında arama yapabılırız

Donguler

```
benimListem=[10,20,30,40,50]
print("Dongu basladı")
for numara in benimListem:
    print(numara)
print("Dongu bitti")
```

cıktı olarak dongu basladı 10 20 30 40 50 ve dongu bıttı yı verır

```
yeniListe=[1,2,3,4,5,6]
for rakam in yeniListe:
   if rakam % 2==0 :
     print(str(rakam) +"  2'ye kalansız bölünür")  2 ye kalansız bolunen lıste elemanlarını lısteledı
```

stringlerle kullanılır

```
yeniString="hilal tabak"
for harf in yeniString:
  print(harf) yazarsak tum harfleri alt alta yazdırır
```

tuple ile kullanımı

```
benimTuple=(1,2,3,4,5)
for eleman in benimTuple:
   print(eleman-10)
```

-9 -8 -7 -6 -5 gibi elemanları yazdırır

koordinatListesi=[(10,2,15.2),(32.4,16.2),(40.2,20.2)]

for eleman in koordinatListesi:

  print(eleman) yaprsak listeler

for(x,y) in koordinatListesi :

print(y) yaparsak ilk degerlerini verir 15.2 16.2 20.2 gibi

benimGaripListem=[(1,2,3),(4,5,6),(7,8,9)]

for (x,y,z) in benimGaripListem:

  print(z) de ise son elemanı verır

sozluklerle calısabılırz

benimSozluk={"muz":150,"portakal":250,"elma":400}

benimSozluk.items() yaprsak

bize ayri ayri listeye sokar yani ([('muz',150),('portakal',250),('elma',400)]('muz',150)]) gibi

type nı ararsak dict_items olarak verir

for (anahtar,deger) in benimSozluk.items():

  print(deger) deresk sadce degerlerı yazdırır yanı 150 200 400 u

Contınіue Break Pass

break

belli bir kosulda donguden cıkmak ıstersek kullanırız

benimListem=[5,10,15,20,25,30]

for numara in benimListem:

  if numara==10:

   break

  print(numara)

5 ve 10 u verır 15e esıt oldugunda donguden cıkar

contınue

benimListem=[5,10,15,20,25,30]

for numara in benimListem:

  if numara==10:

   continue

  print(numara)

cıktı olarak forun ıcındekı ıslemler yapılamdı donguden anlık cıktı ve donguye devam ettı 5 10 20 25 30 yazdırdı 15 i yazdırmadı

pass

for numara in benimListem:

  pass

eger break ya da contınıue yazzaydık hata verırdı ancak daha sonra yazacaksak ya da bos bırakıcaksak vs pass yazıp dkkate alma anlamındadır

Donguler

while

bir kosul tuttugu surece devam et anlamındadır

ornek

x=0

while x <10:

  print(x)

  x = x+1 seklınde bır söz dızımı vardır

x=x+1 yapmazzak donguden cıkamaz ve sonsuza kadar gıder

listeden eleman cıkarmak pop(3) listeye eleman eklemek append(3) unutma

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

while 3 in benimListem: 3 benimListede varken

  print("3hala listede")

  benimListe.pop()

3defa print eder

ornek

numara=0

while numara <5:

 if numara==4:

  break

 print(numara)

 numara=numara+1

cıktı olarak 0 1 2 3 verir 4 de donguden cıkar

ornek

yeniDegisken=0

```
 while yeniDegisken<15:
     print("yeniDegiskenin guncel degeri: "+ str(yeniDegisken))
     yeniDegisken=yeniDegisken+1
```

bunun baska bır yoluda vardır

str ye cevırmemız lazım yenıdDegıskenı Yoksa toplama ıslemı yapamaz

ınt ıle str toplanamaz cunku

2.yoluda daha cok kullanırız

```
 yeniDegisken =0
while yeniDegisken <15:
   print(f"yeniDegiskenin guncel degeri :   {yeniDegisken} ")
   yeniDegisken=yeniDegisken+1
```

f format manasındadır

strye cevırmeden basına f koyup suslu parantezz ıcıne degıskenı yazabılıyoruz

yeniDegiskenin guncel degeri : 0

yeniDegiskenin guncel degeri : 1 seklınde 15e kadar gıder 15 dahıl degıl

Veri Bilimi İcin

benimListem=[1,2,3,4,5] seklınde yazıp ardından

```
 for numara in BenimListem :
     print(numara) yazıp ılerlıyorduk daha kolya bır yoluda vardır
```

Range (aralık -genıslık)

range(15) yazarsak 0-15 aralıgını alır 15 dahıl degıl

list(range(25)) yaparsak 0 dan 25e kadar bır lıste olusturur

ornek

```
 for numara in list(range(15)):
   print(numara*5)
```

yaparsak 0 5 10 15 seklınde 70 e kadar gıden numaralrı yazar cıktı olarak

range(stop) seklınde tek bır sayı yazarsak o sayıya 0dan gıden bır aralık belırtır

range(baslangıc,bıtıs,atlama)

range(baslangıc,bitis)

list(range(5,21,4)) yaparsak cıktı olarak [5,9,13,17] alırız prınt edersek

bitisi yazmaz

Enumerate

index=0

for numara in list(range(5,15)):

  print(f" guncel numara: {numara} guncel index: {index} ")

  index=index+1

guncel numara: 5 guncel ındex: 0

guncel numara:6 guncel index:1 seklında ılerler ındex 9 a kadar

seklında yapmaktansa dah kolay bır yol vardır

for eleman in enumerate (list(range(5,15))):

   print(eleman)

(0,5) (1,6) gibi elemanları verır elemanını type ı tuple dır

enumerate daha kolay kullanımdır

for (index,numara) in enumerate(list(range(5,15))):

print(numara) 5den 15 e kadar verir numaraa yerıne ındex yazarsak 0 dan 10 a kadar degerlerı verır

Range (rasgele)

rasgele bır deger atamak ıcın bır kutuphaneden yararlanırız

from random import randint import derken bır seyı ıcerı alıyoruz dahıl edıyoruz

randint(baslangıc,bitis) degerlerı arasında rasgele bır deger dondururr bıze

randint(0,100) yazarsak 0 -100 arasında rasgele deger atar

yeniListe=list(range(0,10)) yaparsak bır lıste olusturur [0,1,2,3,4,5,6,7,8,9] seklında

yeniListe[randint[0,9]] dersek rasgele bır deger verır

dizilerle cok kolay kullanılır

shuffle karıstırmak demektır

from random import shuffle

shuffle(yeniListe) dersek lıstemızın elemanlarını karısık bır sekılde dızer [3,5,7,9 vb]

zip (sıkıstırmak bır araya getırme anlamlarındadır)

yemekListesi=["muz", "ananas","elma"]

kalorıListesi=[100,200,300]

gunListsi=["pazartesi","sali","carsamba"]

eger bunu boyle yaparsak bıze ındekslerıne gore bır lıste ye cevırır

list(zip(yemekListesi,kaloriListesi,gunListesi)) yaparsak print ettigimizde

[('muz',100,'pazartsi'],('ananas',200,'sali'),('elma',300,'carsamba')] seklında cıktı verır

ziplenmisListe=list(zip(yemekListesi,kaloriListesi,gunListesi))

for eleman in ziplenmisListe:

print(type(eleman))

yaparsak tupple verır turunu bıze

Listeler İleri Seviye

listeOrnegi=[]

benimString="atıl samancıoglu"

for harf in benimString:

listeOrnegi.append(harf)

print(listeOrnegi) dersek tum harfleri bosluk dahıl yazdırır bunun dah kolay yolu vardır

yeniString="atıl samancıoglu"

yeniListeOrnegi=[eleman for in yeniString]

print(yeniListeOrnegi] yazarsak yıne aynı cıktıyı verır ve tum harfleri boslukda dahıl olmak uzere yazdırır

ikinciListeOrnegi=[numara*5 for numara in list(range(0,10))] bu lıstedekı her elemanı numara degıskenıne atadım ve 5 ile carparak ıkıncıLısteOrnegıne ekledım prınt edersem cıktı olarak [0,5,10,15,20,25,30,35,40,45] seklınde bır cıktı verır

Fonksiyonlar ve Metotlar

pratıkte aynı ısleme yarar

fonksıyonu calıstırmak ıcın () koymamız gerekır

kendımızde olusturabılırız ya da olusturulmus fonksıyonları kullanabılırız

fonksyıon yazmak ıcın def kulanılır

def ilkFonksiyon():

print("ilk Fonksıyonum") bosluk bırakması kod blogu vardır kendıne ozel suslu parantez kullanılmaz

fonksyıon cagırıldıgında kullanılır

parantez kullanmak cagırmak execute etmek anlamındadır

ilkFonksiyon() dedıgımızde fonkyıon cagırılır

kodları duzenlı yazmaya yarar

bir defa yazdıkdan sonra ıstedıgımız yerde istedigimiz kez cagırabılırız

icerisine gırdı alıp cıktı verebılır yanı gırdı-cıktı ıslemlerı yapabılır

Fonksıyonların Gırdı Ciktı Islemlerı (input & return)

def merhabaDunya():

print("merhabadunya")

kullanıcıdan da arguman alabılırırz ornegın:

```
def merhabaDunya(yazdirilacakİsim):
  print("merhaba")
  print(yazdirilacakİsim)
```

eger sadece merhabaDunya() yazarsak hata alırız bır deger gırmedıgımız ıcın o yuzden bu sekılde kullanmalıyız

merhbaDunya("hilal") gibi cıktı olarak merhaba hilal verir altalta

hatayı onelemk ıcın bır default deger atayabılırız

yanı bır deger vermezzek default degerı alır deger verırsekde o degerı alır ve kullanır

ornek

```
def merhaba(isim= "atil"):
  print("merhaba")
  print(isim)
```

eger boyle yaparsak merhaba() cıktı olarak merhaba atıl verırken

merhaba("phyton") dersek merhaba phyton cıktısını verır

ornek

```
def toplama(numara1,numara2):
    sonuc=numara1+numara2
    print(sonuc)
```

toplama (10,20) dersek cıktı olarak 30 u verır toplama() yaparsak ıse hata verır

3argumanda verebılırız

```
def superToplama(num1,num2,num3):
  print(num1+num2+num3)
```

deyıp superToplama(10,20,30) dersek 60 ı verır cıktı olarak

yeniDegisken=toplama(10,20)

dersek 30 u verır print(yeniDegisken) yaparsak none verır

burada return kullanıp dondurmelı bır sey kullanmadıgımız ıcın degıskene atayp fonksıyonu prınt edemedık

eger dondurmelı retunr kullanıp bır degıskene esıtlyıp cagırırsak none vermez ve ıstedıgımız degerı verır

ornegın

```
def dondurmeliToplama(num1,num2):
  return num1+num2
```

fonksyıonu print(dondurmeliToplama(10,20)) edersek 30u verır

yeniSonuc=dondurmeliToplama(10,20) yapıp yenı sonucu print edersek

print(yeniSonuc) o zaman cıktı olarak none vermez 30u verır type sorarsak ınt verır

ornek

```
def kontrolFonksıyon(s):
  if s=="atıl":
    print("verdiginiz deger atıl")
 else:
    print("verdiginiz deger atıl degıldır")
```

kontrolFonksıyonu("atıl") verdiginiz deger atıl yazdırır print ettırmeden

Args &Kwargs (args arguman demektır)

args * onemlıdır

```
def yenitopla(*args):
   return sum(args)
```

burada ıstedgımız kadar arguman verip fonksyıona katabılırım

yenitopla(10,20,30,40,50,60)

dedıgımızde cıktı olarak 210 verır

yenitopla(10,20)

dersek cıktı olarak 30u verır

```
def benimFonksiyonum(*args):
     print(args)  dersek cıktı olarak (20,30,40) verir
```

type(benimFonksiyonum(20,30,40)) dersek noneType verir ancak print yerine

```
 def benimFonksiyonum(*args):
     return args deyıp print(type(20,30,40)) dersek tupple verır
```

kwargs

burada ** 2tane yıldız kullanırız

```
def ornekFonksiyon(**kwargs):
   print(kwargs) dersek
```

ornekFonksiyon(muz=100, elma=200,ananas=300) dersek

cıktı olarak ('muz': 100,'elma' : 200,'ananas': 300) verir

print(type(ornekFonksiyon(muz=100, elma=200,ananas=300) dersek type olarak noneType verır returne etmedıgımız ıcın

```
def ornekFonksiyon(**kwards):
```

return kwargs

dersek ve ardından print(type((ornekFonksiyon(muz=100, elma=200,ananas=300) dict dondurur

kelime yada baska 2 seyı esleme kullanacaksak kwards kullanırız

ornek

```
def keyWordKontrol(**kwargs):
  if "atil" in kwargs:
      print("atil var")
  else:
      print("atil yok")
```

keyWordKontrol(atil=70,zeynep=50,mehmet=40) dersek atıl var yazdırırr

metot sınıf ıcerısınde kullandıgımız fonksıyonlardır

Onemlı Fonksıyonlar

```
def bolmeIslemi(numara):
       return numara/2
```

bolmeIslemi(20) yaparsak 10.0 verir

```
benimListem=[1,2,3,4,5,6,7,8,9,10]
yeniListe= []
for eleman in benimLisstem:
    yeniListe.append(bolmeIslemi(eleman))
```

print(yeniListe) dersek cıktı olarak [0.5,1.0,1.5,2.0,3.0,3.5,4.0,4.5,5.0] verir

Map haritalamak manasındadrır

map(bolmeIslemi,benimListem) yaparsak prınt edıp bıze hafıızadakı yerını soyler ama bunu

list(map(bolmeIslemi,benimListem)) yapıp print(list) dersk yıne cıktı olarak [0.5,1.0,1.5,2.0,3.0,3.5,4.0,4.5,5.0] verir

bir listeye bir fonksyıonu uygulayıp liste cevirmek ıstersek oldukca kolaylık saglar

baska verı tıplerındede olur

```
def kontrolFonksiyonu (string):
  return "a" in string
```

kontrolFonksiyonu("atıl") yaparsak true doner

ornek

```
stringListesi=["atıl","samancıoglu","atlas","zeynep","ahmet","levent"]
```

list(map(kontrolFonksiyonu,stringListesi)) stringListesine kontrol fonks uyugulayacak ve yenı bır list olusturacak

hilal=list(map(kontrolFonksiyonu,stringListesi))

print(hilal) yaparsak [True, True, True, False, True, False] cıktısı verır

Filter

list(filter(kontrolFonksiyonu,stringListesi))

deyip bır degıskene atayp yazdırırsak ["atıl",samancıoglu","atlas","ahmet"] verır cıktı olarak

sadece true donenlerı lısteler

Lambda

tek satırda ısleri halletmemize olanak saglar

carpma= lambda numara: numara *3

carpma degıskenımıze numara adında bır degısken alıp 3ile carpıp gerıye deger dondurecegım anlamına gelır

print(carpma(10)) dersek 30 verir

list(map(lambda numara : numara*4,ornekListesi)) yapıp yazdırırsak [40,80,120] verır cıktı olarak

fonksyıonları tek satırda yazarız

Kapsam(scope)

olusturdugumuz degıskenlerın nerede calısacagını ve nereden ulasabılcegımızı gosterır.

numara=20

def carpma(rakam):

  numara=20

  return numara*rakam

carpma(5) dersek 50 alırken print(numara)Dersek 20 alırız

Local- Enclosing- Global- Built İN

orn

```
 benimAdim="atil"

#global

def benmimFonsiyonum():

    benimAdim="mahmut"

    #enclosing
```

```
    def icFonksiyon():
        benimAdim="ayse"
burada ayse olmasaydı benim fonksiyonum() yazdırsaydık mahmut
yazdıracaktı
        #local
        print(benimAdim)
    icFonksiyon()
benmimFonsiyonum()
#ayse yazdırdı
print(benimAdim)
#atıl yazdırdı
y=10
def yeniFonksiyon(y):
   print(y)
   y=5
   print(y)
   return y
yeniFonksiyon(3)
#3 ve 5 cıktısını verır cunku ılk parametre aldıgı
# degerı yazdırdı daha sonra 5 esıtlendı ve 5ı yazdırdı
#globale bakmaya ıhtıyac yoktu
```

orn

```
y=10
def yeniFonksiyon(y):
    print(y)
    y=5
    print(y)
    return y
burada ust taraftakı y yı degıstırmek ıcın yaptık
y=yeniFonksiyon(3)
print(y)
#y return degeerıne esıt oldu en son cıktı olarak 3 5  verır
```

orn

ust tarafta tanımlanan seyın degerını degıstırmek ıstıyorsak global y kullanmalıyız

```
y=10
def ornekFonksiyon():
     global y
     y=5
     print(y)

ornekFonksiyon()
print(y)
```

ıkısıde 5 cıktısını verır cunku global yazdıgımızda ust taraftakı y nın degerını de degıstırmıs olduk

```
def hesapla(a,b,islem):
    if islem == "+":
        return (str(a)+" + " + str(b)+ " = " +str (a+b))
    if islem == "-":
        return (str(a)+" - " +str(b)+ " = " + str(a-b))
    if islem == "*":
        return (str(a)+" * " +str(b)+ " = " +str (a*b))
    if islem == "/":
        return (str(a)+" / " +str(b)+ " = " +str (a/b))
while True:
    a=int(input("İlk sayiyi giriniz: "))
    b=int(input("İkinci sayiyi giriniz: "))
    islem=input("islemlerden birinin seciniz: * / - + ")
    print(hesapla(a,b,islem))
```

print etmezzek yazdırmaz returndekılerı

```
islem=input("islemlerden birinin seciniz: * / - + ")
```

true oldugu surece sonsuza kadar dondurur

sadece hesaplayı cagırırsak prınt olmadıgı ıcın sonucu ekrana yazdırmaz return ıcıne prınt ekleyebılırız str den once

```
    if islem not in " +- /* ":
        return "lutfen gecerli islem girin" basa bunu
```

eklersek ıstemedıgımız ıslem oldugu ıcın hata mesajı verecek ve basten ısteyecek degerlerımızı

try except

trydakı kodları dener hata alırsa except komutundakı kod blogunu calıstırır

try catch komutları gıbı cdakı

```
def hesapla(a,b,islem):
 if islem not in " +- /* ":
        return "lutfen gecerli islem girin"
    if islem == "+":
        return (str(a)+" + " + str(b)+ " = " +str (a+b))
    if islem == "-":
        return (str(a)+" - " +str(b)+ " = " + str(a-b))
    if islem == "*":
        return (str(a)+" * " +str(b)+ " = " +str (a*b))
    if islem == "/":
        return (str(a)+" / " +str(b)+ " = " +str (a/b))
while True:
   try:
     a=int(input("İlk sayiyi giriniz: "))
    b=int(input("İkinci sayiyi giriniz: "))
    islem=input("islemlerden birinin seciniz: * / - + ")
    print(hesapla(a,b,islem))
  except:
      print("lutfen sayıları daha duzgun giriniz: ")
```

hatalı gırersek sayıları excepte duserız

OOP - NESNE YONELIMLI PROGRAMLAMA

instance & attribute

sınıf ıcerısındekı fonksıyonlara metoto dıyoruz

Genelde sınıfların ılk harfı buyuk yazılır

fonksıyonlar gıbı tanımlanır seklı

__init__ tanımlı bır fonksyıon zate ınıtıalize den yanı baslatma anlamına gelmektedır

```
class SuperKahraman():
```

```
    def __init__ (self):
     print("İnit cagirildi")
```

self i eklemezzek hata verır self argumanını almalı

superman=SuperKahraman() yaparsak dırek yoksa hataverır

intit cagırıldı cıktısı verır

batman=SuperKahraman() yaparsak yıne ınıt cagırıldı cıktısı verır

```
class SuperKahraman():
    def __init__ (self,isimInput,yasInput,meslekInput):
        self.isim=isimInput
        self.yas=yasInput
        self.meslek=meslekInput
        print("İnit cagirildi")
```

```
superman=SuperKahraman() yaparsak hata verır cunku arguman vermedık
3 tane parametre alıyor ınıt self dzaten default tanımlı
```

```
yanı su sekılde olmalıı
  superman=SuperKahraman("superman",30,"gazateci")
```

```
cıktısı ınıt cagırıldı seklınde olur
```

buradakı self ısım yas ve mesleklerın tanımlanmasına yardım eder

superman.isim deyip cagırabılırım yada sonrasında atanan degerı degıstırebılırım superman.isim="hilala" gib

> Bir sınıf oluşturduğunuzda, genellikle __init__ metodu tanımlayarak yeni nesnelerin nasıl oluşturulacağını belirtirsiniz.
>
> __init__ metodu içerisinde self anahtar kelimesini kullanarak yeni nesnenin özelliklerini ayarlarsınız.

```
superman=SuperKahraman() burada aslında nesne olustururuz ve
parantez ıcıne yazdıgımız argumanlar bızım nesnenın ozellıkleri
olur
```

> initi nesneyı baslatır ve aynı zamanda gırdıleri yonetır
>
> degiskenler otomatık saglanır ve her zaman ılk degısken olarak yerlestırılır

ornek

```
class Character:
    def __init__(self,health,damage,speed):
        self.health=health
```

```
        self.damage=damage
        self.speed=speed
    def double_speed(self):
        self.speed*=2
warrior=Character(100,50,10)
ninja=Character(80,40,40)
print(f"warriod speed: {warrior.speed}")
print(f"ninja speed: {ninja.speed}")
warrior.double_speed()
print(f"warriod speed: {warrior.speed}")
print(f"ninja speed: {ninja.speed}")
```

default deger atama

```
class Kopek():
    yilCarpani=7
    def __init__(self,yas=5):
        self.yas=yas

    def insanYasiniHesapla(self):
        return self.yas*self.yilcarpani

benimKopek=Kopek()
print(benimKopek.yas)
yaparsak cıktı 5 verir ancak
benimKoper=Kopek(3)
deyıp yasını prınt edersek 3 verır overrıde etmıs oluruz
print(benimKopek.insanYasiniHesapla()) yaparsak 35 verir cıktı
olarak
    self.yılcarpanı yerıne kopek.yılcarpanı da dıyebılırız
```

inheritance - mıras alma

```
class Hayvan():
    def __init__(self):
        print("hayvan sinifi init cagirildi")
```

```
    def method1(self):
        print("hayvan sinifi method1  cagirildi")
    def method2(self):
        print("hayvan sinifi method2 cagirildi")


class Kedi(Hayvan):
    def __init__(self):
        Hayvan.__init__(self)
        print("kedi sinifi init cagirildi")
    def miyavla(self):
        print("miyav")
```

//Overrride

```
    def method1(self):
        print("kedi sinifi method1 calistirildi")
digerKedi=Kedi()
digerKedi.method1()
hayvan sinifi init cagirildi
kedi sinifi init cagirildi
kedi sinifi method1 calistirildi cıktısı verır
```

polymorphism

aynı ısımdekı mettoların vb farklı amaca hızmet etmesı gıbıdır

```
class Elma ():
    def __init__(self,isim):
        self.isim=isim
    def bilgiVer(self):
        return  self.isim+ "100 kaloridir"


class Muz ():
    def __init__(self,isim):
        self.isim=isim
    def bilgiVer(self):
```

```
        return  self.isim+ "150 kaloridir"

elma=Elma("elma")

print(elma.bilgiVer())

muz=Muz("muz")

print(muz.bilgiVer())

meyveListesi=[elma,muz]

#buradakı elma ve muz tırnak ısaretı ıcınde yazılmadıgı ıcın

#muz ve elma sınıfındakı objelerdır

for meyve in meyveListesi:

    print(meyve.bilgiVer())

#her ikisinde de bilgi ver metotu valıstı ve kendıne has
ozellıklerıyle

#calıstı

def bilgiAl(meyve):

    print(meyve.bilgiVer())

bilgiAl(elma)
```

Ozel metotlar

```
class Meyve():

    def __init__(self,isim,kalori):

        self.isim=isim

        self.kalori=kalori

    def __str__(self):

        return f"{self.isim} su kadar kaloriye sahiptir :
{self.isim}"

    def __len__(self):

        return self.kalori

muz=Meyve("muz",150)

print(muz.kalori)

print(muz)

#hafızadakı yerını yazıyor ya da len fonksıyonunu yazınca hata
verıyor

#ozel metotları kullanarak cozebılırız

#str tanımlayarak yazdırdım muzu
```

```
elma=Meyve("elma",200)
print(len(elma))
```

Hatalar

try & except & else& finally

```
while True:
    try:
        benimInt=int(input("Numaranizi giriniz:"))
    except:
        print("Lütfen gercekten numara giriniz ")
        continue
    else:
        print("tesekkurler")
        break
    finally:
        print("finally cagirildi")
Numaranizi giriniz:atıl
Lütfen gercekten numara giriniz
finally cagirildi
Numaranizi giriniz:51
tesekkurler
finally cagirildi
```

OOP - NESNE YONELIMLİ PROGRAMLAMA

__init__ initialize yanı olusturmaktan gelır ve yapıcı metot olarak dusunebılırız.Yani sinif içerisinde bir metot olusturdugumuzda geleneksel olarak ilk argumanı alırlar.

Bu argumana self derız dah sonra eklemek ıstedımız argumanları verebılırız.

Selfin anlamı classtan turettıgımız herhangı bır objeyi temsil eder .

init metotuna constructor yanı yapıcı metot olarak tanımlayabılırız

obje turetıldıgı zaman otomatık olarak calısır

ınıt ıcındekı parametreleri obje olusturduktan sonra belırtmemız geerekıyor ama self ıcın bır parametre atamamamız gerekmez cunku zaten self olusturulan objeyı temsıl eder

self yerıne baska adlarda kullanabılırız ornegın thıs ama kod butunlugu acısından self kullanmak daha ıyıdır

```
class Person:
    def __init__(self,name,year):
      self.Name=name
      self.Year=year
      print("init metotu calisti.")


p1=Person("Ali",1990)
p2=Person("yagmur",1995)
print(f"name : {p1.Name} year: {p1.Year}")
print(f"name : {p2.Name} year: {p2.Year}")
```

Sag taraftakı name argumanın adı gelecek argumanın adını tutar sol taraftakı ise cagırırken kullandıgımız ad ıkısıde aynı olabılır ama neyın ne oldugunu anlatmak ıcın boyle kullandım

///bard//

Python'da **init** metotu, bir sınıftan nesne oluşturulduğunda otomatik olarak çağrılan özel bir metottur. Bu metota **yapıcı** veya **başlatıcı** da denir.

init metodunun görevleri:

Nesnenin **özniteliklerini** (değişkenlerini) tanımlamak ve başlangıç değerlerini atamak.

Nesnenin **davranışlarını** belirleyen **metotlari** tanımlamak.

Nesnenin **durumunu** başlatmak için gerekli işlemleri yapmak.

init metodunun parametreleri:

self: Yeni oluşturulan nesnenin referansını tutar. Bu parametre, metodun içinde nesnenin özniteliklerine ve metotlara erişmek için kullanılır.

Diğer parametreler: Nesnenin oluşturulması sırasında kullanıcının girebileceği ek parametreler.

init metodu ile ilgili önemli noktalar:

init metotu, **her sınıfta opsiyoneldir**. Bir sınıfın **init** metotu olmayabilir.

init metotu, **bir kere** çağrılır. Nesne oluşturulduktan sonra tekrar çağrılmaz.

init metotu, **herhangi bir değer döndürmez**.

init metotunda, **self** parametresinin **ilk parametre** olması zorunludur.

init metotu ile ilgili örnekler:

Bir öğrenci sınıfı için **init** metodunda öğrencinin adını, soyadını ve numarasını tanımlayabilirsiniz.

Bir araba sınıfı için **init** metodunda arabanın markasını, modelini ve rengini tanımlayabilirsiniz.

Bir geometrik şekil sınıfı için **init** metodunda şeklin kenar uzunluklarını veya yarıçapını tanımlayabilirsiniz.

```
class Kisi:
    def __init__(self, isim, soyisim, dogum_tarihi):
        self.ad = isim
        self.soyad = soyisim
        self.yas = yas_hesapla(dogum_tarihi)
def selamla(self):
        print(f"Merhaba, ben {self.ad} {self.soyad}.")
kisi1 = Kisi("Ahmet", "Yılmaz", "1997-01-01")
print(kisi1.ad)  # "Ahmet"
print(kisi1.soyad)  # "Yılmaz"
print(kisi1.yas)  # 25
```

Bu örnekte:

init metodunun parametreleri: isim, soyisim, dogum_tarihi

Nesnenin öznitelikleri: ad, soyad, yas

Sağdaki ve soldaki değişken adları farklıdır.

Hangi yöntemin kullanılacağı, kodlama tarzına ve kişisel tercihe bağlıdır.

Önemli olan:

Kodun okunabilir ve anlaşılır olması

Değişken adlarının anlamlı olması

her zaman kullanmayacagınız attrıbuteleri class attrıbutes olarak tanımlyabılırsınız

her zaman kullanacagımız ozellıkleri object attrıbutes olarak tanımlayabılırsınız

attributes(ozellık -nitelik)

Sınıf Nitelikleri (Class Attributes) Nedir?

Sınıf nitelikleri, bir **sınıfın gövdesinde** doğrudan tanımlanan ve **sınıfın tüm örnekleri tarafından paylaşılan** niteliklerdir. Sınıftan oluşturulan tüm nesneler arasında tutarlı kalan **sınıf düzeyi verileri** depolamak için bir yol sağlarlar.

Sınıf Niteliklerinin Özellikleri:

Tanımlama ve Paylaşım:

Herhangi bir yöntemin (yapıcı __init__ dahil) dışında, sınıf gövdesinin içinde tanımlanır.

Hem sınıf adı (SınıfAdı.sınıf_niteliği) hem de sınıfın örnekleri (nesne_adı.sınıf_niteliği) kullanılarak erişilebilir.

Bir sınıf niteliğinin değeri, sınıfın tüm örnekleri tarafından paylaşılır. Bir örnek üzerinden değeri değiştirmek, diğer tüm örnekler için de değeri değiştirir.

Kullanım Alanları:

Sabitleri Depolamak: Sınıf nitelikleri, dönüştürme faktörleri, sınıf işlevselliğine özgü matematiksel değerler veya yapılandırma ayarları gibi sınıfla ilgili sabitleri depolamak için idealdir.

Varsayılan Değerler: Nesne nitelikleri (örnek nitelikleri) tipik olarak aynı varsayılan değeri paylaşıyorsa, bu değerler için sınıf nitelikleri tanımlayabilirsiniz. Gerekirse nesne oluşturulurken bu değerler geçersiz kılınabilir.

Sınıf Düzeyi Verileri Takip Etmek: Bazı durumlarda, tek tek nesnelerden ziyade sınıfın kendisiyle ilişkili verileri takip etmek isteyebilirsiniz. Örneğin, bir sınıf niteliği kullanarak bir sınıftan oluşturulan nesnelerin sayısını takip edebilirsiniz.

Örnek:

Python

```
class Daire
PI = 3.14159  # Sınıf niteliği (sabit
def __init__(self, yarıçap):
self.yarıçap = yarıçap  # Örnek niteliği

def alan(self):
return Daire.PI * self.yarıçap**2

daire1 = Daire(5
daire2 = Daire(3
print(daire1.alan())  # 78.53975
print(daire2.alan())  # 28.25997

$ınıf niteliğini (PI) değiştirmek her iki daireyi de etkiler
Daire.PI = 3.14  # PI değerini değiştir

print(daire1.alan())  # 78.5  (Yeni PI ile alan yeniden hesaplanır
print(daire2.alan())  # 28.26  (Yeni PI ile alan yeniden hesaplanır
```

Use code with caution.

content_copy

Önemli Noktalar:

Bir sınıf niteliği, bir örnek üzerinden değiştirildiğinde tüm örnekleri etkilemesine rağmen, açıklık açısından sınıf adı üzerinden doğrudan değiştirmek genellikle önerilir.

Sınıf niteliklerini bilinçli bir şekilde kullanın. Aşırı kullanım, nesneler arasında sıkı bir bağlantıya yol açabilir ve kodu daha az esnek hale getirebilir.

```
#methods

    def intro():
```

```
        print("hello there") sadece bu sekılde yapıp p1.intro()
dersk hata alırız ınstance methods yanı objelere hızmet edecek bır
metottur
```

self i arguman olarak vermelıyız

```
class Person:
    #class attributes(class ozellıkleri -nitelikleri)
    addres="no information"

    #constructor(yapıcı metot)
    def __init__(self,name,year):
        #object attributes
        self.name=name
        self.year=year

     #instance methods
    def intro(self):
        print("hello there. I am "+ self.name)
    def calculateAge(self):
        return 2019-self.year



#object (instance)
p1=Person(name="ali",year=1990)
p2=Person(name="yagmur",year=1995)
#accesing object attributes
""" yorum satiri yapmak için 3 tirnak isareti kullan
print(f"p1: name : {p1.name} year : {p1.year} adress: {p1.addres}")
print(f"p2: name : {p2.name} year : {p2.year} adress: {p2.addres}")
"""
p1.intro()
p2.intro()
#intro metotundakı self gelcek objeyı temsıl edıyor
```

```
print(f"adim : {p1.name} ve yasim: {p1.calculateAge()}")
print(f"adim : {p1.name} veyasim: {p2.calculateAge()}")
```

Ornek

```
class Circle:
    #class object attributes
    pi=3.14
    def __init__(self,yaricap=1):
        self.yaricap=yaricap

    #methods
    def cevre_hesapla(self):
        return 2 * self.pi * self.yaricap

    def alan_hesapla(self):
        return self. pi * (self.yaricap**2)

c1=Circle() # yaricapi1 alır
c2=Circle(5)
print(f" c1 : alan : {c1.alan_hesapla()} ve cevresi :
{c1.cevre_hesapla()}")
print(f" c2 : alan : {c2.alan_hesapla()} ve cevresi :
{c2.cevre_hesapla()}")
```

Kalıtım ınheritance : mıras alma olarak da kullanılabılır

```
class Person():
    def __init__(self):
        print("person crated")

class Student(Person):
    def __init__(self):
        print("student created")
p1=Person()
s1=Student()
```

sekılde bırakırsam studentın ınıtı personun ınıtını ezer overrıde eder eger bunu ıstemıyorsak

```
 Person.__init__(self) eklersek studentın ınıtıın altına personunda ınıtı calısmıs olur

class Person():
    def __init__(self):
        print("person crated")


class Student(Person):
    def __init__(self):
        Person.__init__(self)
        print("student created")
p1=Person()
s1=Student() olursa person created daha sonra student created cıktısı verır
```

return kullanmanın avantajları:

Fonksiyonu diğer fonksiyonlarla birlikte kullanmayı kolaylaştırır.

Aynı ısımdekı metot temel sınıftakı metotu ezer override etmek denıyor

```
 Person.__init__(self,fname,lname) yı temel sınıftakı yapıcı metotu kullanmak ıcın kullandıgımızı bılıyoruz bunun yerıne su alternatıfıde ekleyebılıriz:

super().__init__(fname,lname)

#Inheritance (kalitim) : miras alma
"""
Person => name,lastname,age, eat(),run(),drink()

Student (Person) , Teacher(Person)

Animal => Dog(Animal),Cat(Animal)

"""

class Person():
    def __init__(self,fname,lname):
        self.firstName=fname
        self.lastName=lname
        print("person crated")
    def who_am_i(self):
```

```
        print("I am a person")
    def eating (self):
        print("I am eating")


class Student(Person):
    def __init__(self,fname,lname,number):
        Person.__init__(self,fname,lname)
        self.studentNumber=number
        print("student created")

    #override
    def who_am_i(self):
        print("i am a student")

    def sayHello(self):
        print("hello i am astudnet")
class Teacher(Person):
    def __init__(self,fname,lname,branch):
        super().__init__(fname,lname)
        self.branch=branch
    def who_am_i(self):
        print(f"i am a  {self.branch} teacher")
```

ornek

```
p1=Person("ali","yilmaz")
s1=Student("cinar","turan",123456)
t1=Teacher("hilal","tabak","math")
print(p1.firstName+ " " +p1.lastName)
print(s1.firstName+ " " +s1.lastName+" "+str(s1.studentNumber))
t1.who_am_i()
p1.who_am_i()
s1.who_am_i()
```

```
p1.eating()
s1.eating()
s1.sayHello()


mylist=[1,2,3]
"""
myString="my string"
print(len(mylist))
print(len(myString))
print(type(mylist))
print(type(myString))
"""
class Movie():
    def __init__(self,title,director,duration):
        self.title=title
        self.director=director
        self.duration=duration
        print("movie objesi olusturuldu")



m=Movie("film adi","yonetmen adi","film suresi")
#print(len(m))  yaparsak movie objesı ıcın len metotunun olmadıgını belırtır
#print(type(m)) hata verır yanı bu metotları kullanamam
print(str(m))
print(str(mylist))
yaparsak mylıstı yazdırırken m nın hafızadakı yerını yazar sadece
def __str__(self):
        return f"{self.title} by {self.director}" eklersek movıe classına o zaman fılmın adı by fılmın yonetmenı seklınde cıktı verır

 def __len__(self):
        return self.duration    eklersek len metotunu da kullanabılırız
```

```
print(len(m)) yaparsak 120 cıktısını verır

def __del__(self):

        print("film objesi silindi") yapıp print(m) yaparsak

movie objesi olusturuldu

film adi by yonetmen adi

film objesi silindi cıktısı verır
```

metot- fonksıyon farkı ve nedır?

ikiside aynı amacla kullanılır ama kullanım yerlerı farklılık gosterır

metot:metotları barındıran kod butunune class yanı sınıf derız.class ve classtan turetılen nesneden ulasabılırz

fonksyıonlar: bellı amaclar ıcın olusturdugumuz kod bloklarıdır metotlarla benzer ısleri yapar.class ıcerısınde tanımlamıyoruz

Python'da Metot ve Fonksiyon: Farklılıklar ve Benzerlikler

Metot ve fonksiyon, Python'da kod bloklarını tanımlamak ve tekrar kullanmak için kullanılan yapılardır. Her ikisi de kodun daha organize ve modüler hale getirilmesine yardımcı olur.

Farklılıklar:

Tanımlama:

Metotlar: Bir **sınıfın** içinde tanımlanır.

Fonksiyonlar: Sınıfın **dışında** tanımlanır.

Bağlılık:

Metotlar: Bir **nesneye bağlıdır** ve o nesne üzerinde çalışır.

Fonksiyonlar: Herhangi bir nesneye bağlı **değildir**.

Erişim:

Metotlar: Nesnenin **özellikleri** gibi erişilebilir.

Fonksiyonlar: Koddaki **herhangi bir yerden** çağrılabilir.

Parametreler:

Metotlar: İlk parametresi **self** veya **cls** anahtar kelimesiyle başlar.

Fonksiyonlar: Herhangi bir özel parametresi yoktur.

Benzerlikler:

Kod bloğu: Her ikisi de bir kod bloğu içerir.

Parametreler: Her ikisi de parametre alabilir.

Dönüş değeri: Her ikisi de bir değer döndürebilir.

Örnek:

Python

```
class Daire
pi = 3.14

def __init__(self, yarıçap):
self.yarıçap = yarıçap

def cevre_hesapla(self):
return 2 * self.pi * self.yarıçap

def alan_hesapla(yarıçap):
return Daire.pi * yarıçap**2

daire1 = Daire(5
print(daire1.cevre_hesapla())  # 31.4
print(alan_hesapla(5))  # 78.5
```

Bu örnekte:

cevre_hesapla bir **metot**tur, çünkü bir sınıfın içinde tanımlanmıştır ve bir nesne üzerinde çalışır.

alan_hesapla bir **fonksiyondur**, çünkü sınıfın dışında tanımlanmıştır ve herhangi bir nesneye bağlı değildir.

Hangi yapıyı kullanacağınız:

Bir kod bloğunun **tekrar kullanılabilir** olması ve **sınıfın işlevselliğiyle** ilişkili olması durumunda **metot** kullanın.

Bir kod bloğunun **tekrar kullanılabilir** olması ancak **sınıfa bağlı olmaması** durumunda **fonksiyon** kullanın.

Ek Bilgiler:

Metotlar, **nesneye yönelik programlamada** önemli bir rol oynar.

Fonksiyonlar, **genel amaçlı kod bloklarını** tanımlamak için kullanılır.

ornek

```
def add(a,b,c=0):
    return sum((a,b,c))
print(add(10,20))
print(add(10,20,30))
#yerine
print("*          *")
```

```
def add(*params):
    #print(params) #tuple olarak tum elemanları verır
    #return sum((params))
    sum=0
    for n in params:
        sum=sum+n

    return sum
print(add(10,20,30,40))
print(add(10,20,30))

def displayUser(**args):
    for key,value in args.items():
        print("{} is {}" .format(key,value))


displayUser(name="cinar",age=1,city="istanbul")
displayUser(name="ada",age=12,city="izmir")
displayUser(name="yigit",age=15,city="kastomonu")
```

virsual stdioda terminal acmak ıcın ctrl ve " tuslarına basınız

py dosya adı yazarsak ya da python dosya adını yazarsak da kod calısır

dosya olusturmak ıcın mkdir dosya adı

dosyanın ıcını ormek ıcın ls

dosya ıcıne dosya acmak ıcın mkdır dosya adı yıne

konsoldan cıkmak ıcında ctrl ; e basınız

yorum satırı yapmak ıcın ctrl k +c ye basarsanız ard arda secılem yer yorum satırı olur ,kaldırmak ıcın ıse ctrl k u seklında ard artda basınız

print(type(x),type(Y) ,type(z)) seklınd tek satırda yazabılırız

may var may-var 2myvar gecersız kullanımdır

camel case = myVariable

pascal case = MyVariable

snake name= my_variable

x,y,z = "portakal","muz", "kıraz"

print(x,y,z) seklınde kullanım olabılır

a=b=c="kivi"

print(a,b,c) olabılır

meyveler=["elma","muz","kiraz"]

d,e,f=meyveler

print(d,e,f)

yaptıgımızda hafleri meyveler dızısındekı elemanlarla eslestırır

x=5

y=10

print("sonuc",x+y) sekında kullanabılırız ya da su sekılde print("sonuc="+str(x+y)) print(f " {x + y}") seklınde de olur

str ve int yanyana yazmak ıstersek

global olarak tanımlarsak fonk dısındada kullanabılırız

def fonk():

  global x

  x="bulutlu"

fonk()

print("hava"+x) hava bulutlu cıktısı verır

x ı en ustte x="yagmurlu" dersek bulutlu yu yazar yagmur yerıne bulutlu olur x

Complex (Karmaşık Sayılar) Karmaşık sayılar sanal kısım olarak "j" ile yazılır: Örnek Complex: x = 3+5j y = 5j z = -5j print(type(x)) print(type(y)) print(type(z))

String'ler (Arrays) Dizeler Diğer birçok popüler programlama dili gibi, Python'daki dizeler de unicode karakterleri temsil eden bayt dizileridir. Ancak Python'un bir karakter veri türü yoktur, tek bir karakter yalnızca 1 uzunluğunda bir dizedir. String öğelerine erişmek için köşeli parantezler kullanılabilir. Örnek Karakteri 1 konumunda alın (ilk karakterin 0 konumuna sahip olduğunu unutmayın): a = "Merhaba Dünya!" print(a[1])

Boşluğu Kaldırma Boşluk, asıl metinden önceki ve/veya sonraki boşluktur ve çoğu zaman bu boşluğu kaldırmak istersiniz. Örnek strip() yöntemi, başlangıçtaki veya sondaki tüm boşlukları kaldırır: a = " Merhaba Trakya! " print(a.strip())

String'i Değişiklik Yapma Örnek replace() yöntemi, bir dizeyi başka bir dizeyle değiştirir: a = "Merhaba Trakya!" print(a.replace("M", "N"))

Bölünmüş string (dize) split() yöntemi, belirtilen ayırıcı arasındaki metnin liste öğeleri haline geldiği bir liste döndürür. Örnek split() yöntemi, ayırıcının örneklerini bulursa dizeyi alt dizelere böler: a = "Merhaba, Tekirdağ!" print(a.split(",")) # ['Merhaba', 'Tekirdağ!'] değerini geri döndürür

Ancak, format() yöntemini kullanarak dizeleri ve sayıları birleştirebiliriz! format() yöntemi iletilen bağımsız değişkenleri alır, biçimlendirir ve {} yer tutucularının bulunduğu dizeye yerleştirir:

String Biçimlendirme Python Değişkenleri bölümünde öğrendiğimiz gibi, dizeleri ve sayıları şu şekilde birleştiremeyiz: Örnek Yas = 28 txt = "Benim adım İpek, yaşım: " +age print(txt)

Örnek Dizelere sayılar eklemek için format() yöntemini kullanın: yas = 28 txt = " Benim adım İpek, yaşım: {}" print(txt.format(yas)) format() yöntemi sınırsız sayıda argüman alır ve ilgili yer tutuculara yerleştirilir:

Örnek miktar = 3 itemno = 567 fiyat = 49.95 siparisim = "{} adet {} numaralı üründen {} TL tutarında istiyorum." print(siparisim.format(miktar, itemno, fiyat)) Bağımsız değişkenlerin doğru yer tutuculara yerleştirildiğinden emin olmak için {0} dizin numaralarını kullanabilirsiniz: Örnek miktar = 3 itemno = 567 fiyat = 49.95 siparisim = "{1} numaralı üründen {0} adet için {2} TL ödemeliyim." print(siparisim.format(miktar, itemno, fiyat))

Bazı Değerler Yanlış Aslında (), [], {}, "", 0 sayısı ve Yok değeri gibi boş değerler dışında False olarak değerlendirilen çok fazla değer yoktur. Ve elbette, False değeri False olarak değerlendirilir. Çoğu Değer Doğru Bir tür içeriğe sahipse, hemen hemen her değer True olarak değerlendirilir. Boş dizeler dışında herhangi bir dize True'dur. 0 dışında herhangi bir sayı True'dur. Boş olanlar dışında herhangi bir liste, tanımlama grubu, küme ve sözlük True'dur.

Python ayrıca, bir nesnenin belirli bir veri türünde olup olmadığını belirlemek için kullanılabilen isinstance() işlevi gibi bir boole değeri döndüren birçok yerleşik işleve sahiptir:

// yaparsak bolme ıslemınekı kalanı bıze vermez

sayı negatıf oldugunda asagı yuvarlar

-15//4 derse k -4 der

10//3 dersek 3 alırken -10//3 dersek -4 sonucu alırız

format kullanımı

name ="cinar"

surname="turan"

print("my name is {} {} " .format(name,surname)) dersek my name ıs cınar yazdırır

suslu paranteze ındeks numarası verebılırız yada

print("my name is {s} {n} " .format(n=name,s=surname)) de yapabiliriz

print("my name is {} {} and ı am {} years old   " .format(name,surname,,age)) dıyeblırız tur donusumune gerek yok age yerıne "36" da yazabılırız

result=200/700

print("the result is {r:1.4}" .format(r=result)) dersek vırgulden once kac karakterlık bosluk ıcın 1 vırgul sonrası kac karakter ıcın 4 u yazdık

ifadeyi tersten yazdırmak ıcın [::-1]

s="12345"*5

print(s[::5]) dersek 11111 yazar

title() her harfın bas harfını buyuk yapar

capatılıza () dersek ılk kelımeyı buyuk yapar

strip() bas ve sonrakı boslukları karakterlerı sıler

lstrip() soldakı boslukları sıler rsplit() dersk sagdakı boslukları sıler

split() cumleyı bosluklara gore dızı halıne sokar

message="hello there .my name is hilal"

message=message.split()

kelıme dızısıne cevırır

bırlestırmek ıcın ıse joın metotu kullanılır

message=" " .joın(message) bırlestırırken araya bosluk ekler

find ile aramak ıstedgımız kelımeyı ararırz ındex doner ve kelımenın baslangıc ındeksını verır bıze

eksı deger verırse yoktur anlamındadir

startswith("h") h ile mi baslıyor

endswith("h") ile bitiyormu

replace("hilal","pelin") hilal yerine pelin ekler replaceleri ard arda ekleyebılırız

center (100) 100 karakterlık bır konteıner olusturur ve mesajımızı ekler bosluk ekler 100 karakter sag ve sola

message=message.center(50,"*") deresek sak ce sola 50 karakterlık yıldız ekler

indekx("h") dersk ındeks numarası doner rindeks deresk sagdan ındeks numarası verır rfınd ıle aynı

listeye append ile eklenır insert ındekse ggore ekler

pop() en sondakı sılınır ındeks verebiliriz remove ile karakter verebiliriz

sort sayusal olarak sıralanır letters.sort dersek alfaetık sıralanır

reverse ters cevrılır

tek bır eleman atamya ızı nyok tuplede

tuple lerı toplanabılır

dictıonarıyler suslu parantez kullanılır

```
users={
```

```
    "sadikturan":{
      "age":36,
      "email":"sadik@gmail.com",
      "addres":"kocaeli",
      "phone":"7890"
    },
    "hilaltabak":{
      "age":50,
      "email":"hilalk@gmail.com",
      "addres":"ist",
      "phone":"123456"
    }
}
print(users["sadikturan"]["age"])
```

set suslu parantezle yapılır

ındeks ılgılerıyle ulasılamaz ındekslenemez set

elemanlara ulasmak ıcın for kullanılır

y ada sralayamayız

add ile ekleyebiliriz

update ile liste ekleyebılırz

aaynı elemanı ekleyemeyız aynı elemandan bırden fazla olmaz

remove ile sılebılırız

discard ile de sılebılırz pop ıle yaparsak herhangı bır elemanı sıler

values=1,2,3,4,5

x,y,*z=values

print(x,y,z) dersek ilk iki x ve y ye gerı kalanlar lıste halınde z ye gıder

addres karsılastırması yaprasak is kullanılırz

```
#range kullanımı
# for item in range(0,100,10):
#   print(item)
```

```
# print(list(range(5,100,10)))

#enumerate

# index=0

# greeting="hello there"

# for letter in greeting:

#   print(f"letter : {index} letter: {letter}")

#   #  print(f"letter : {index} letter: {greeting [index]}")
seklınde de kullanabılırsınız

#   index +=1

#bu sekılde ındex numarası tanımlamadan kullanım seklıde vardır
enumerate kullanarak

#greeting="hello"

# for item in enumerate(greeting):

#   print(item)

 # for index,letter in enumerate(greeting):

  #yada item yazarsak ve print edersek

 # print(f"index: {index} letter : {letter}")

 #sadece ıtem yazarsak indeks ve harf ı lıstteler halında verır

#birden fazla listeyi eslestirir ve yeni bir list dondurur

# list1=[1,2,3,4,5]

# list2=["a","b","c","d","e"]

# list3=[100,200,300,400,500]

#for a,b,c in zip(list1,list2,list3):

#   print(a,b,c)


# print(list(zip(list1,list2)))

#numbers=[]

# for x in range(10):

#   numbers.append(x)


# print(numbers)


#bu sekılde kullanım harıcında daha kolay kullanımıda vardır
```

```
#numbers=[x for x in range(10)]
# print(numbers)
#for x in range(10):
#   print(x**2)

# numbers=[x**2 for x in range (10)]
# print(numbers)
#numbers=[x*x for x in range(10) if x %3==0]
# print(numbers)
#myString="hello askom"
# myList=[]
#for lettter in myString:
#   myList.append(lettter)
# print(myList)
#myList=[letter*3 for letter in myString]
# print(myList)
#years=[1983,1999,2008,1956,1986]
# ages=[2019-year for year in years]
# print(ages)
#gelecek x degerlerını buradan ayarlıyoruz
# results=["CİFT" if x%2==0 else "tek" for x in range(1,10)]
# print(results)
#result=[]
# for x in range(3):
#   for y in range(3):
#     result.append((x,y))

# print(result)
# #bu ıslemı kısaca yapma ıslemı ıse
#numbers=[(x,y) for x in range(3) for y in range(3)]
# print(numbers)
import random
```

```
sayi=random.randint(1,100)
can =int(input("kac hak kullanmak istersiniz: "))
hak=can
sayac=0
while hak>0:
  hak-=1
  sayac+=1
  tahmin=int(input("tahmin ettginiz sayiyi giriniz : "))
if sayi==tahmin:
    print(f"{sayac}. defada bildiniz.Toplam puaniniz: {100- (100/
can)*(sayac-1)}")
    break
  elif sayi> tahmin:
    print("yukari")
  else:
    print("asagi")

  if hak==0:
    print(f"hakkiniz bitti.tutulan sayi : {sayi}")
#def yasHesapla(dogumYili):
#   return 2019-dogumYili
#def EmekliligeKacYilKaldi(dogumYili,isim):
#   """
#   hello askom
#   """
#   yas=yasHesapla(dogumYili)
#   emeklilik=65-yas

#   if emeklilik>0:
#     print(f"{isim} insani emekliliginize {emeklilik} yil kaldi")

#   else:
#     print("zaten emekli olmussun moruk")
```

```
# EmekliligeKacYilKaldi(1990,"hilal")
# print(help(EmekliligeKacYilKaldi))
# def changeName(n):
#   n="ada"

# name="hilal"
#changeName(name)
# print(name)
#def change(n):
#   n[0]="istanbul"

# sehirler=["ankara","izmir"]
# # n=sehirler[:] #slicing
#n[0]="istanbul"
# change(sehirler[:])
#print(sehirler)
#def add(a,b):
#   return sum((a,b))
#print(add(10,20))
# gibi 2den fazla parametre verırsek hata verır
print(add(10,20,30))
#cyı 0 a esıt yapıp c yı prınt ederek 3.parametreyıde ekleyebılırız
# def add1(*params):
#   return sum((params))
# print(add1(10,20,30))
#sum kullanmak ıstemezzek eger
# def add1(*params):
#   sum=0
#   for n in params:
```

```
#     sum=sum+n
#   return sum
#**args yada **params dıyebılırız dıctıonary kullanılanacagını gostermek ıcın
#onemlı olan 2 yıldız kullanmak
# def displayUser(**args):
#   for key,value in args.items():
#       print("{} is {}".format(key,value))




# displayUser(name="cinar",age=12,city="istanbul")
# displayUser(name="ada",age=2,city="kocaeli",phone="123456")
# displayUser(name="yigit",age=14,city="ankara",phone="45678")
#def myfunc(a,b,*args,**kwargs):
#   print(a)
#   print(b)
#   print(args)
#   print(kwargs)
#myfunc(10,20,30,40,50,key1="value1",key2="value2")
#for kullanımı
#numbers dekı her elemanı sırayla num a esıtle ve prınt etttıj
# numbers=[1,2,3,4]
# for a in numbers:
#   print(a)
#tuple=[(1,2),(1,3),(3,5),(5,7)]
#for a in tuple:
#   print(a)
# d={"k1":1,"k2":2,"k3":3}
#for item in d.items():
#     print(item)
```

```
#   # for item in d: yaparsak sadece key leri verir
#for key,value in d.items():
#     print(value)
#def fonk(kelime,sayi):
#print(kelime*sayi)
#fonk("merhaba\n",10)
# def fonk(*params):
#   liste=[]

#   for param in params:
#     liste.append(param)

#return liste
#result=fonk(10,20,30,"hello")
# print(result)
#1:10
# 2:30
# def asalSayilariBul(sayi1,sayi2):
#   for sayi in range(sayi1,sayi2+1):
#     if sayi>1:
#       for i in range(2,sayi):
#         if(sayi % i ==0):
#           break
#       else:
#         print(sayi)


# sayi1=int(input("1.sayiyi giriniz: "))
# sayi2=int(input("1.sayiyi giriniz: "))
# asalSayilariBul(sayi1,sayi2)
# def tamBolenleriBul(sayi):
#   tamBolenler=[]
```

```
#   for i in range(2,sayi):
#     if (sayi% i ==0):
#       tamBolenler.append(i)

#   return tamBolenler
#print(tamBolenleriBul(50))
#def square(num): return num**2
# square=lambda num: num**2
# numbers=[1,3,5,9,10,4]
#result=list(map(square,numbers))
# # print(result)
# #list yada forla kullan map ı yoksa adresı verır
# # for item in map(square,numbers):
# #   print(item)

# # square=lambda num: num**2 yaptıgımız ısleme ısım de
verebılıyoruz
# # result=square(3)
# def check_event(num): return num%2==0
#result=list(filter(lambda num :num%2 ==0,numbers))
#print(result)
#banka uygulaması
SadikHesap={
  "ad":"Sadik Turan",
  "hesapNo":"12345678",
  "bakiye":3000,
  "ekHesap":2000
}
AliHesap={
  "ad":"Ali Turan",
  "hesapNo":"12345678",
  "bakiye":2000,
```

```
  "ekHesap":1000
}
def paraCek(hesap,miktar):
  print(f"Merhaba {hesap['ad']}")

  if (hesap['bakiye']>= miktar):
    hesap['bakiye']-= miktar
    print("paranizi alabilirsiniz ")
  else:
    toplam=hesap['bakiye']+hesap['ekHesap']

    if(toplam>= miktar):
      ekHesapKullanimi=input("ek hesap kullanilsin mi (e/h)")

      if ekHesapKullanimi=="e":



        ekHesapKullanilacakMiktar=miktar-hesap['bakiye']
        hesap['bakiye']=0
        hesap['ekHesap']-=ekHesapKullanilacakMiktar


        print("paranizi alabilirsiniz")
      else:
        print(f"{hesap['hesapNo']} nolu hesabinizda
{hesap['bakiye']} bulunmaktadir")
    else:
       print("bakiye yetersiz")



paraCek(SadikHesap,3000)
paraCek(SadikHesap,2000)
```

cls Nedir?

Python sınıf yöntemlerinde cls, **sınıfın kendisine** atıfta bulunan özel bir parametredir. Bireysel nesne örnekleri üzerinde değil, sınıfın kendisi üzerinde çalışan bir yöntemdir. Sıklıkla sınıf niteliklerini (sınıfın tüm örnekleri tarafından paylaşılan değişkenler) oluşturmak veya değiştirmek için kullanılan fonksiyonları tanımlamak için kullanılır.

cls Ne Zaman Kullanılır?

Bir fonksiyonu @classmethod dekoratörü ile süsleyerek bir sınıf yöntemi olduğunu belirtirsiniz. Sınıf yöntemi içinde cls, sınıfın kendisine atıfta bulunur ve size sınıf seviyesi niteliklerine erişme ve bunları değiştirme imkanı verir.

```
class Çalışan
zam_miktari = 1.04  # Sınıf değişkeni (tüm Çalışan nesneleri
tarafından paylaşılır
@classmethod
def zam_miktari_ayarla(cls, miktar):
cls.zam_miktari = miktar  # cls kullanarak sınıf değişkenini ayarla

calisan1 = Çalışan("Ahmet", "Yılmaz", 50000
calisan2 = Çalışan("Ayşe", "Demir", 60000
#Başlangıç zam miktarını yazdır
print(f"Varsayılan zam miktarı: {Çalışan.zam_miktari}")
#Sınıf yöntemini kullanarak zam miktarını değiştir
Çalışan.zam_miktari_ayarla(1.05
#Güncellenmiş zam miktarını yazdır (tüm çalışanlara uygulanır
print(f"Yeni zam miktarı: {Çalışan.zam_miktari}")
```

Kodu dikkatli kullanın.

content_copy

Bu örnekte:

- zam_miktari, tüm Çalışan nesneleri tarafından paylaşılan bir sınıf değişkenidir.
- zam_miktari_ayarla, @classmethod ile süslenmiş bir sınıf yöntemidir.
- zam_miktari_ayarla içinde cls, Çalışan sınıfına atıfta bulunur ve cls.zam_miktari = miktar kullanarak zam_miktariyi değiştirmenize izin verir.

Anahtar Noktalar:

- Sınıf seviyesi niteliklere erişmek ve bunları değiştirmek için sınıf yöntemlerinde cls kullanın.
- cls belirli bir nesne örneğine atıfta bulunmaz.
- Sınıf davranışını ve niteliklerini yönetmek için güçlü bir araçtır.

Ek Hususlar:

- cls genellikle sınıf yöntemlerinde kullanılır, ancak normal yöntemlerde de kullanılabilir (daha az yaygın olsa da). Bu durumda, metodun çağrıldığı nesne örneğine atıfta bulunur.

Bir yöntem içinde sınıf niteliklerini değiştirmeniz gerekmiyorsa, cls'ye ihtiyacınız olmayabilir.

Python'da Statik Metotlar

Statik metotlar, **sınıfın veya nesnelerin durumuna** erişmeden çalışan özel bir metot türüdür. Sınıfın veya nesnelerin bilgilerine erişemedikleri için **bağımsız fonksiyonlar** gibi davranırlar.

Statik Metotların Kullanım Alanları:

Yardımcı fonksiyonlar: Sınıfın veya nesnelerin durumuyla ilgisi olmayan matematiksel işlemler veya veri dönüştürme işlemleri gibi fonksiyonlar için kullanılabilir.

Fabrika fonksiyonları: Belirli koşullara bağlı olarak farklı nesneler oluşturmak için kullanılabilir.

Sınıf sabitlerini tanımlamak: Sınıfın tüm örnekleri tarafından paylaşılan sabit değerleri tanımlamak için kullanılabilir.

Statik Metot Tanımlama:

Statik metotlar, @staticmethod dekoratörü ile süslenerek tanımlanır.

Python

```
class Araba
def __init__(self, marka, model):
self.marka = marka
self.model = model

@staticmethod
def km_mil_cevir(km):
return km * 0.621371

araba1 = Araba("Fiat", "Egea")
#statik metotu nesne üzerinden çağırmak mümkündür
print(araba1.km_mil_cevir(100))  # 62.1371

#sınıf üzerinden de çağrılabilir
print(Araba.km_mil_cevir(100))  # 62.1371
```

Kodu dikkatli kullanın.

content_copy

Statik Metotların Özellikleri:

Sınıfın veya nesnelerin **self** parametresine erişemezler.

Sınıfın veya nesnelerin **niteliklerine** erişemezler.

Bağımsız fonksiyonlar gibi davranırlar.

Sınıfın veya nesnelerin **durumunu değiştirmezler**.

Statik Metotların Avantajları:

Kod tekrarını azaltır.

Daha modüler ve okunabilir kod sağlar.

Sınıfın veya nesnelerin durumuyla ilgisi olmayan fonksiyonlar için daha uygun bir yapı sunar.

Statik Metotların Dezavantajları:

Sınıfın veya nesnelerin durumuna erişemedikleri için bazı durumlarda daha az esnek olabilirler.

Sınıfın veya nesnelerin **özel bilgilerini** kullanamamaları nedeniyle bazı işlemler için uygun olmayabilirler.

Statik Metot Kullanmaya Karar Vermeden Önce:

Fonksiyonun **sınıfın veya nesnelerin durumuna** erişip erişmediğini göz önünde bulundurun.

Fonksiyonun **bağımsız bir fonksiyon** olarak kullanılabilir olup olmadığını değerlendirin.

Kodun **okunabilirliği** ve **modülerliği** üzerindeki etkisini göz önünde bulundurun.

Statik metotlar, Python'da kodunuzu daha modüler ve okunabilir hale getirmek için kullanabileceğiniz güçlü bir araçtır. Doğru şekilde kullanıldığında, kod tekrarını azaltmanıza ve daha esnek ve bakımı kolay kod yazmanıza yardımcı olabilirler.

Uygulamamıza mudahale edebılmek ıcın parcalara ayırıyoruz ve her parcaya modul dıyoruz

her parca bır py uzantılı oparcaya denk gelmektedır.hre bır module farklı gorevler verebılmekteyız ve bu moduller arasında baglantılar olabılır

kendi hazırladıgımız moduller

hazır moduller

a)standart kutuphane modulleri:python gelistiricileri tarafından gelıstırılmıs

b)ucuncu sahıs modulleri

Dosya İşlemleri

Dosya acmak ve olusturmak için open() fonksıyonu kullanılır

Kullanımı:open(dosya_adi,dosya_erisme_modu)

dosya_erisme_modu => dosyayı hangı amacla actıgımızı belirtir

"w" write yazma modu.Dosya konumunda olusturur.yazdıgımız bılgıler daha sonra yazdıgımız bılgıler tarafından sılnır .son ekledıgımız bılgı dosya ıcersınde yer alır.

dosyayı konumda olusturur .dosya ıcerıgını sıler ve yenıden ekleme yapar

"a" append ekleme.Dosya konumda yoksa olusturur.daha once var olan verılerın uzerıne ekleme yapmıs oluruz .kodu tekrar tekrar calıstırısak o kadar ekleme yapar

"x" create olusturma.Dosya zaten varsa hata verir. actıktan sonra 2kez calıstırısak hata verır

"r" okuma .varsayılan.Dosya konumda yoksa hata verir.exception firlatır

Python'da dekoratör fonksiyonlar, **var olan fonksiyonlara yeni özellikler eklemek** için kullanılan güçlü bir araçtır. Bu eklemeler, fonksiyon çağrılmadan önce, sonra veya her ikisi de olmak üzere gerçekleştirilebilir.

Neden Dekoratör Fonksiyonları Kullanırız?

Dekoratör fonksiyonlarının kullanımı çeşitli avantajlar sağlar:

Kod Tekrarını Önler: Aynı işlemleri birden fazla yerde yazmak yerine, tek bir dekoratör fonksiyonu yazabilir ve bu fonksiyonu ihtiyaç duyduğunuz diğer fonksiyonlara uygulayabilirsiniz. Bu yaklaşım kodun daha temiz ve bakımı kolay olmasını sağlar.

Kodu Daha Okunabilir Hale Getirir: Dekoratör fonksiyonları sayesinde ortak kod parçaları asıl fonksiyonun gövdesinden ayrılır ve ayrı bir dekoratör fonksiyonuna taşınır. Bu sayede asıl fonksiyonun ne yaptığını anlamak daha kolay hale gelir.

Yeni Davranışlar Ekler: Dekoratör fonksiyonları, var olan fonksiyonlara yeni yetenekler kazandırmak için idealdir. Örneğin, bir fonksiyonu giriş yetkisi kontrolünden geçirmek, çalışma süresini ölçmek veya önbelleğe alma gibi işlemleri dekoratör fonksiyonlar aracılığıyla gerçekleştirebilirsiniz.

plt.legend() fonksiyonu da Matplotlib'te önemli bir rol oynar ve aşağıdaki amaçlar için kullanılır:

1. Farklı Çizgileri veya İşaretleyicileri Açıklamak:

Bir grafikte birden fazla çizgi veya işaretleyici kullanıldığında, plt.legend() fonksiyonu, her birinin hangi veri kümesini temsil ettiğini gösteren bir efsane oluşturmak için kullanılır. Bu, grafiğinizi daha kolay okunabilir ve anlaşılır hale getirir.

2. Efsanenin Konumunu ve Biçimini Kontrol Etmek:

plt.legend() fonksiyonu, efsanenin konumunu (örneğin, "loc" parametresiyle) ve biçimini (örneğin, "fontsize" parametresiyle) kontrol etmenize olanak tanır. Bu, grafiğinizin genel görünümünü ve profesyonelliğini özelleştirmenize yardımcı olur.

3. Efsanedeki Girişleri Gizlemek veya Göstermek:

plt.legend() fonksiyonu, belirli çizgileri veya işaretleyicileri efsaneden gizlemenize veya göstermenize olanak tanır. Bu, grafiğinizi daha karmaşık hale getiren veri kümelerini gizlemek için kullanılabilir.

TENSORFLOW

Sigmoid Fonksiyonu

0 ile 1 arasında deger alır

genelde sınıflandırma problemlerımızde ıse yarar

Tahn(hıperbolık tanjant)fonksıyonu

-1 ile 1 arasında deger alır

negatıf degerlerle daha genıs bır kapsam saglar ve genelde sınıflandırma operasyonlarında kullanılır

ReLU(rectified linear unit)

0 ile sonsuz arasında deger alır

derın ogrenmede sıklıkla karsımıza cıkar

verılen rakam negatıf ıse 0 ı verır eger negatıf degılse verılen degerı alır

Linear Fonksıyonlar

f(x)=x

sonsuz deger alabılır fakat non-lınear olmaması sebebıyle modellerde sorunlara yol acabılır

VERİ

OGRENME VERİSİ

TEST VERİSİ

Maliyet Fonksıyonu(error -hata fonksıonu):Gercek verıden ne kadar uzaktayız bunu olcer.Bu degerı mınımalıze etmeye (dusurmeye) calısırız.Ne kadar dusukse o kadar ıyı calısır.

Gradient Descent:Bir fonksıyonun mınımumunu bulmak ıcın kullandıgımız optımızasyon fonksyıonudur(Optimizasyon fonksiyonu, bir problemin çözümünü en iyi hale getiren bir fonksiyondur. Bu fonksiyon, belirli bir kriteri minimize veya maksimize ederek çalışır.)Maliyet fonksyıonunu mınımıze etmek ıcın kullanırız.

Optimize etmek, bir sistemi veya süreci en iyi hale getirmek için değişiklikler yapmak anlamına gelir.

Asagı ınerken ne kadar hızlı veya yavas ınebılecegımızı secebılırız(step sıze).Bu aslında ogrenme sevıyemızle aynı anlama gelır.Mınımıma hızlı ulasmak ıcın optımızasyonlar kullanabılırız

Populer secenekler:RMSProp,Adam

Kodların Açıklaması:

1. MinMaxScaler'ı İçe Aktarma:

from sklearn.preprocessing import MinMaxScaler satırı, scikit-learn kütüphanesinden MinMaxScaler sınıfını içe aktarır. Bu sınıf, verileri 0 ile 1 aralığına ölçeklendirmek için kullanılır.

2. MinMaxScaler Objesi Oluşturma:

scaler = MinMaxScaler() satırı, MinMaxScaler sınıfından bir nesne oluşturur. Bu nesne, verilerin ölçeklendirilmesi için kullanılacak modeldir.

3. Eğitim Setine Sığdırma:

scaler.fit(x_train) satırı, ölçekleyici modelini eğitim setine sığdırır. Bu işlem modelin, verilerin minimum ve maksimum değerlerini öğrenmesini sağlar.

4. Eğitim Setini Ölçeklendirme:

x_train = scaler.transform(x_train) satırı, ölçekleyici modelini kullanarak eğitim setini ölçeklendirir. Bu işlem sonucu eğitim setindeki değerler 0 ile 1 aralığına düşürülür.

5. Test Setini Ölçeklendirme:

x_test = scaler.transform(x_test) satırı, aynı ölçekleyici modelini kullanarak test setini de ölçeklendirir. Böylece test seti de aynı aralığa ölçeklenmiş olur.

6. Test Setinin Yazdırılması:

print(x_test) satırı, ölçeklendirilmiş test setini ekrana yazdırır. Bu sayede ölçeklendirme işleminin sonucunu inceleyebilirsiniz.

Özet:

MinMaxScaler, verileri 0 ile 1 arasında ölçeklendiren bir sınıftır.

Önce model eğitim setine sığdırılır, sonra bu model kullanılarak hem eğitim hem de test setleri ölçeklendirilir.

Bu işlem, makine öğrenmesi modellerinin performansını artırır.