Değişkenler

- Sayı ile başlayamaz
- Değişken isminde boşluk olamaz
- İçinde matematiksel operatörler yada Python ifadeleri olamaz.

metot	anlamı
count()	kaç adet var?
find()	varmı? index no?
len()	karakter/eleman sayısı
split()	Parçalara ayırır
replace()	Değiştirir,Yerine koyar
upper() lower()	Harf büyüt küçült
type()	Değişken tipini verir
isinstance(değişken,float)	Belirtilen tipteyse True döner

Kullanıcıdan Bilgi Almak

```
alınanbilgi= input("Adınız nedir?")
```

*Alınan bilgi string tipindedir.

format() Fonksiyonu

```
print("Verilen değer parantezlere {} yerleşir".format("Değer"))
```

int(),float(),str() ile değişkenleri birbirine çevirebilirsin.

Koleksiyonlar

```
liste= []   liste= list()
```

```
demet= ()   demet= tuple()
```

```
sözlük= {}   sözlük= dict()
```

```
küme= set()
```

Liste

liste += [“eleman1,eleman2”]	Eleman ekle
liste.append()	Eleman ekle
len(liste)	Eleman sayısı
liste[0]	İlk elemanı getir
liste.upper()	Harfleri büyütür
liste.pop()	Son elemanı siler
liste.pop(2)	İkinci pozisyondakini siler
liste.insert(2,”eleman”)	İkinci pozisyona ekler
liste.remove(“eleman”)	Belirli bir elemanı siler
liste.clear()	Listeyi temizler
liste.index(12)	12 Sayısının pozisyonunu verir
liste.copy()	Bağımsız kopya oluşturur

liste.reverse()	Tersine çevirir
liste.sort(reverse = True)	Sıraya dizer
liste.extend(ek_liste)	Listeye liste ekler

Sözlükler

```
sozluk_1 = {"anahtar1":"deger1","anahtar2":"deger2","anahtar3":"deger3"}
print(sozluk_1["anahtar1"])
sozluk_1["anahtar1"]= "Yeni Değer"
sozluk_1["yeni_anahtar"]=  "Yeni Değer"
```

pop(“anahtar adı”)	Belirtilen elemanı siler
sözlük.get(“anahtar”,”Bulunamadı”)	Bulunamadı mesajı verebiliriz
sözlük.keys()	Tüm anahtarları getirir
sözlük.items()	(anahtar1,değer1),(anahtar2,değer2)
sözlük.popitem()	Siler, (anahtar,değer) döndürür
sözlük.values()	Değer kısımlarını döndürür

Demetler

```
demet = ()
demet_sayılar = (1,2,3,4,5)
tek_elemanlı_demet = (1,)
```

count()	Demetin eleman sayısını getirir
index()	Belirtilen elemanın pozisyon numarası

```
demet_1 = (1,2,3,4,5)
demet_2 = ("a","b","c")
print(demet_1  + demet_2)
```

Kümeler

```
boş_küme = set()
küme= {1,2,3}
```

Kümeye liste,sözlük,başka bir küme eklenmez. Küme içinde sadece demet olabilir.

küme.add(“eleman”)	Kümeye eleman ekler
sebzeler.update({"soğan","Sarımsak"})	Çoklu eleman eklemek
sebzeler.clear()	Kümeyi temizler
sebzeler.remove(“Soğan”)	Kümeden eleman siler
sebzeler.discard(”Patates”)	Siler, eleman yoksa hata vermez.
sayilar.pop()	Rastgele siler.

Birleşim Kümesi

```
a_kumesi = {1,2,3,4,5,6,7,8}
b_kumesi = {6,7,8,9,10,11,12}
print(a_kumesi | b_kumesi)
```

Kesişim Kümesi

```
a_kumesi = {1,2,3,4,5,6,7,8}
b_kumesi = {6,7,8,9,10,11,12}
print(a_kumesi & b_kumesi)
```

Kümelerde Fark

```
a_kumesi = {1,2,3,4,5,6,7,8}
b_kumesi = {6,7,8,9,10,11,12}
print(a_kumesi-b_kumesi)
```

*Kümelerde eleman tekrarlanmaz

*list(), dict(), tuple(), set() fonksiyonlarıyla birbirine dönüşürler.

Akış Kontrolleri

Matematiksel Operatörler

Operatör	Açıklama	Örnek
+	Toplama	3 + 3 = 6
-	Çıkarma	5 - 2 = 3
*	Çarpma	3 * 3 = 9
/	Bölme	19 / 3 = 6.3
//	Taban Bölme	19 // 3 = 6
**	Üs Alma	2 ** 3 = 8
%	Mod Alma	12 % 5 =2

Kıyaslama Operatörleri

Operatör	Anlamı
==	Eşittir
<=	Küçüktür veya eşittir
>=	Büyüktür veya eşittir
!=	Eşit Değildir
<	Küçüktür

>	Büyüktür

Boolean Örnekleri

```
True == True
>>>True
```

```
True == False
>>>False
```

```
True is True
>>>True
```

```
True is not True
>>>False
```

Mantıksal Operatörler

Operatör	**Anlamı**
and	İki şart da sağlanmalı
or	Biri sağlansa yeterli
not	Tam tersine çevirir

if Kullanımı

```
if yaş>18:
    print("Ehliyet alabilir.")
```

else Kullanımı

```
if yaş>18:
    print("Ehliyet alabilir.")
else:
    print("Henüz ehliyet alamaz..")
```

elif Kullanımı

```
if yaş<18:
    print("Henüz ehliyet alamaz..")
elif yaş>100:
    print("Ehliyet almak için fazla yaşlısınız.")
else:
    print("Ehliyet alabilir.")
```

if Sorgusu Pratik Yöntem

```
durum = ("Hız limiti aşıldı" if hiziniz>hiz_limiti else "Hızınız normal")
```

Döngüler

while Döngüsü

```
sayaç= 10
while sayaç>=0:
    print("Geri sayım: ",sayaç)
    sayaç-=1
```

break

```
sayaç= 10
while sayaç>=0:
    print("Geri sayım: ",sayaç)
    if sayaç == 5:
        break #Döngüden çıkar
    sayaç-=1
```

continue

```
sayaç= 10
while sayaç>=0:
    print("Geri sayım: ",sayaç)
    if sayaç == 5:
        continue #sayaç 5 ise sayacı 1 azaltmadan döngünün başına döner
    sayaç-=1
```

for Döngüsü

```
sayılar= range(0,10)
for sayı in sayılar:
    print(sayı)
```

enumerate

```
aylar= ["Ocak","Şubat","Mart"]
for sıra_no,ay in enumerate(aylar):
    print("{}.{}".format(sıra_no,ay))
```

Fonksiyonlar

```
def fonk_adı(param_1="varsayılan", param_2=None):
    #yapılacaklar

fonk_adı(1,2) #fonksiyon adını yazarak çağırılır.
```

```
def fonksiyon(a,b):
    print(a)
    print(b)

fonksiyon(1,2)
```

Pozisyonal Arguman

```
fonksiyon(1,2)
```

Anahtar Kelimeli Argüman

```
fonksiyon(param1=1,param2="Merhaba")
```

*Değer döndürme ifadesi: return

*Fonksiyona parametre olarak liste gönderirsen ve fonksiyonun aldığı parametre değerini değiştirirsen gönderilen orijinal listenin değeri de değişir.

*args

```
def çoklu_değerler(*args):
    print(args)

çoklu_değerler(1,3,5,7,"Ali","Mehmet",55.6)
```

Çıktı: (1, 3, 5, 7, 'Ali', 'Mehmet', 55.6)

**kwargs

```
def çoklu_kelimesel_fonksiyon(**kwargs):
    print(kwargs)

çoklu_kelimesel_fonksiyon(isim="Namık",soyad="Kamil",yas=25,boy=1.85)
```

Çıktı: {'isim': 'Namık', 'soyad': 'Kamil', 'yas': 25, 'boy': 1.85}

Anonim Fonksiyonlar-Lambda

```
toplama= lambda sayı1,sayı2 :  sayı1 + sayı2
print(toplama(14,20))
```

Çıktı: 34

String Biçimlendirme

```
isim= "Selim"
print("{} isimli öğrenci sınavdan 100 aldı.".format(isim))
```

Çıktı: Selim isimli öğrenci sınavdan 100 aldı.

```
metin = "Birinci: {} İkinci: {} Üçüncü:
{}".format("değer1","değer2","değer3")
print(metin)
```

Çıktı: Birinci: değer1 ikinci: değer2 üçüncü parantez: deger3

```
metin = "Benim adım {2} soyadım {1} boyum {0}dir."
print(metin.format("Kenan","Güney",1.78))
```
Çıktı: Benim adım 1.78 soyadım Güney boyum Kenandir.

Kelime hedefli olarak gönderme

```
metin= "Benim adım {isim} soyadım {soyad} boyum {boy}dir."
print(metin.format(isim="Şaban",soyad="Yılmaz",boy=1.85))
```
Çıktı: Benim adım Şaban soyadım Yılmaz boyum 1.85dir.

format methodu olmadan string biçimlendirme

```
isim = "Vedat"
print("Benim adım %s" % isim)
```
Çıktı: Benim adım Vedat

%s string
%d integer
%f float

```
ücret= 59.90446654333
print("Aldığım şapkanın ücreti: %.2f TL" % ücret)
```
Çıktı: Aldığım şapkanın ücreti: 59.90 TL

*isim[1:].lower() ilk karakter dışındakileri küçük harf yapar.

Değişkenlerin Kapsamları

```
genel_toplam = 12.5
def ürün_ekle():
    global genel_toplam
    genel_toplam += 10
    print(genel_toplam)
```

```
ürün_ekle()
```
Bu şekilde genel kapsamlı değişkeni fonksiyon içinden değiştirmeniz mümkün oluyor.

Hata Yakalama-Kriz Yönetimi

```
try:
    print(2/0)
except:
    print("Belli ki bir hata oldu")
```
Çıktı: Belli ki bir hata oldu

```
try:
    print(2/0)
except ZeroDivisionError:
    print("Sayıyı sıfıra bölemezsin")
```
Çıktı: Sayıyı sıfıra bölemezsin

```
try:
    print(2/0)
except (ZeroDivisionError):
    print("Sayıyı sıfıra bölemezsin")
except IndexError:
    print("Hatalı bir index numarası girdiniz")
except (TypeError,ImportError):
    #Bu şekilde iki durum için de aynı exception bloğu çalışır.
    pass
except:
    print("Exception yazmadığımız bütün diğer hata tiplerinde de burası çalışır")
else:
    #Kullanımı tercihe bağlıdır.Bir hata yoksa çalışır
finally:
    #Kullanımı tercihe bağlıdır.Her şartta, hata olsun olmasın çalışır. Genellikle memoryde kullanılan kaynakları serbest bırakmak için kullanılır.
```

raise

```
raise TypeError
```

Çıktı: TypeError

```
raise MemoryError("Keyfi verdirilmiş bellek hatası")
```

Çıktı: MemoryError: Keyfi verdirilmiş bellek hatası

```
try:
    raise ZeroDivisionError("Bir sayıyı sıfıra bölemezsin anla artık")
except ZeroDivisionError:
    print("Bir hata var")
```

Çıktı: Bir hata var

```
try:
    sayı= int( input("Çift sayı girin"))
    if sayı % 2 ==0:
        pass #Devam et dedik, pas geç
    else:
        raise ValueError("Ben sana sadece çift sayı girebilirsin dedim!")
except ValueError as hata_mesajı:
    print("Hata mesajımız: " + str(hata_mesajı))
```

Üst Düzey Fonksiyonlar

filter() Fonksiyonu

```
def çiftleri_bulma_makinesi(sayı):
    if sayı %2 ==0:
        return True
    else:
        return False

sayılar = [0,1,2,3,4,5,6,7,8,9,10]

filtreden_geçirilmis_çift_sayılar =
filter(çiftleri_bulma_makinesi,sayılar)

print(list(filtreden_geçirilmis_çift_sayılar))
```

Çıktı: [0, 2, 4, 6, 8, 10]

map() Fonksiyonu

```
sayilar = range(0,10)
islenmis_sayilar = map(lambda sayi : sayi +1,sayilar)
list(islenmis_sayilar)
```

Çıktı: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

reduce() Fonksiyonu

```
from functools import reduce
sayilar = range(1,5)
print(reduce(lambda sayi1,sayi2:sayi1*sayi2,sayilar))
```

Çıktı :24

any() Fonksiyonu

En az bir True

```
liste = [False,False,False]
any(liste)
```

Çıktı: False

```
liste = [False,True,False]
any(liste)
```

Çıktı: True

all() fonksiyonları

Hepsi True olmalı.

```
liste = [1233,"Merhaba",123,False]
all(liste)
```

Çıktı: False

```
liste = [1233,"Merhaba",123]
all(liste)
```
Çıktı: True

List Comprehension-Akıllı Listeler

```
liste_1 = [1,2,3,4,5,6,7,8,9,10]
liste_2 = [sayı+1 for sayı in liste_1]
print(liste_2)
```
Çıktı: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

```
liste_1 = [1,2,3,4,5,6,7,8,9,10]
liste_2 = [sayı for sayı in liste_1 if sayı%2==0]
```
Çıktı: [2, 4, 6, 8, 10]

```
liste_1 = [1,2,3,4,5,6,7,8,9,10]
liste_2 = [sayı for sayı in liste_1 if not (sayı==2 or sayı==3)]
```
Çıktı: [1, 4, 5, 6, 7, 8, 9, 10]

Akıllı Listede İç İçe Sorgular

```
sayilar = range(1,100)
sartlara_uyan_sayilar = [sayi for sayi in sayilar if sayi % 3 ==0 if
sayi % 7 ==0 ]
print(sartlara_uyan_sayilar)
```
Çıktı: [21, 42, 63, 84]

Akıllı Listede Else Kullanımı

```
sartlara_uyan_sayilar = ["bölünür" if sayi%6==0 else "bölünmez" for sayi in sayilar]
```

Curried Fonksiyonlar - Taksitli Parametre

```
def birinci_fonksiyon(para1):

    def ikinci_fonksiyon(para2):

        def ucuncu_fonksiyon(para3):

            print(para1,para2,para3)

        return ucuncu_fonksiyon

    return ikinci_fonksiyon

Çağır: birinci_fonksiyon(10)(20)(30)
```

Çıktı: 10 20 30

Nesne Tabanlı Programlama

Sınıf tanımlama aşamaları
Sınıfın adı
Sınıfın Nitelikleri
init fonksiyonu
Örneğin Nitelikleri
Örneğin Metotlar
Sınıfın Metotları
Static Metodlar

```
class Kedi():
    #init fonksiyonu ve örnek nitelikleri
    def __init__(self,isim,yas):
        self.isim = isim
        self.yas = yas
```

Örnekleme

```
Class A():
    pass
örnek = A()
```

Örneğin Niteliklerine Erişmek

```
class Kişi():
    def __init__(self,isim,yas):
        self.isim = isim
        self.yas = yas

Ahmet = Kişi("Ahmet",30)
print(Ahmet.yas)
```

Örneğin Metotları

```
class Araba():
    def __init__(self,renk,hız):
        self.renk = renk
        self.hiz = hız
        self.calisiyor = False

    def calistir(self):
        self.calisiyor = True

kara_simsek = Araba("Siyah",180)
kara_simsek.calistir()
```

Sınıfa ait Nitelikler

```
class Araba():

    uretim_sayisi = 0


print(Araba.uretim_sayisi) #Sınıf adıyla çağrılır
```

Sınıfa Ait Metotlar

```
class Araba():

   uretim_sayisi = 0

   @classmethod
   def uretim_sayisini_artir(cls):
       cls.uretim_sayisi += 1
```

Static Metot

```
class Araba():
    @staticmethod
    def benim_static_metodum():
        print("Ben static metodum")
```

Kalıtım-Inheritence

```
class Robot():
    pass

class AşçıRobot(Robot):
    pass
```

***super() fonksiyonu üst sınıfın fonksiyonuna ulaşabilmemizi sağlar.**

```
class Öğretmenler():
    def __init__(self,isim,soyad,id):
        self.isim = isim
        self.soyad = soyad
        self.id = id

class Kimya_Öğretmenleri(Öğretmenler):
    def __init__(self,isim,soyad,id,labsuresi)
        super().__init__(isim,soyad,id) #üst sınıfın tanımlamasını unutma
        self.labsuresi = labsuresi
```

__str__

```
class Tekne():
    def __str__(self):
        return "Bu bir teknedir"

t1 = Tekne()
print(t1)
```

Çıktı: Bu bir teknedir

Kapsülleme Ve Erişim Belirleyiciler

```
self.__mesajlar = mesajlar
```

Değişkenleri özel yapabildiğimiz gibi metotları da özel yapabiliriz.

```
def __kullaniciyi_sil(self):
    print("Kullanıcı silindi.")
```

Property Dekoratörü(Nitelikleştirici)

```
class Kitap():
    def __init__(self,baslik):
        self.__baslik = baslik

    @property
    def baslik(self):
        return self.__baslik

v1 = Kitap("Abc")
print(v1.baslik)
```

Çıktı: Abc

*Property salt okunurdur

Property .setter dekoratörü

```
class Kitap():
    def __init__(self,baslik):
        self.__baslik = baslik

    @property
    def baslik(self):
        return self.__baslik

    @baslik.setter
    def baslik(self,yeni_baslik):
        self.__baslik = yeni_baslik
        print("Başlık değiştirildi.")

k1 = Kitap("Sefiller")
k1.baslik = "Karamazov Kardeşler"
print(k1.baslik)
```

Çıktı: Karamazov Kardeşler

Property .deleter dekoratörü

```
@baslik.deleter
def baslik(self):
    del self.__baslik
    print("Başlık silindi")

k1 = Kitap("Sefiller")
del k1.baslik
```

Çıktı: Başlık silindi

Modüller ve Modül Import Etmek

```
import modül_adı
```

Modüle Takma İsim Koymak

```
import matematikmodulum. as islem
```

Modülün bir bölümünü import etmek

```
from matematikmodulum import toplama
```

__name__ Niteliği

```
if __name__ == "__main__":
    #Program direk çalışıyor başka program içinde modül olarak değil.
```