Python

Derlenmeden çalıştırılabilen popüler programlama dilidir, böyle dillere script (betik) diller denilir. Derlenen dile örnek olarak c verilebilir. C de yazdığınız bir program parçası önce yerel makine koduna çevirilir, sonra çalıştırılır.

Ödev: a- Başka hangi script dilleri var?
b- Derlenen başka hangi programalama dilleri vardır ?

Python platform bağımsız bir dildir. Yazdığınız programın fazla değişikliğe gitmeden windows, linux ya da mac te çalıştırılıyor olmasıdır.

Python ile neler yapılabilir:

- Web uygulamaları (sunucu tarafı)
- Veri analizi
- Robotik uygulamalar
- Yapay zeka ve makine öğrenimi
- Oyun geliştirme
- Görüntü işleme
- Sistem komut dosyaları oluşturma

gibi birçok yazılım alanlarında kullanılan öğrenilmesi kolay, anlaşılabilir ve daha kısa kodlara sahip bir dildir.

Python Kurulumu

Pythonda program yazmak ve çalıştırmak için çeşitli ortamlar mevcuttur.
İşletim sistemine kurulabildiği gibi online web tabanlı python yorumlayıcı siteler mevcuttur.
Komutları yazmak için bir editör kısıtlaması yoktur. Metin editöründe yazılabildiği gibi, gelişmiş editör (IDE) ler mevcuttur.
https://www.python.org/ adresinden kullanıdığınız işletim sistemine uygun olan dosya indirilip kurabilir.
Python da kısa komutları çalıştırmak için ptyhon komut satırı kullanılabilir, ama çok satırlı program yazılacaksa bir editorde komutları yazıp çalıştırmak daha avantajlıdır. Hataları görüp düzenlemek daha hızlı olacaktır.
Ptyhon komut satırı için işletim sistemi komut satırına “cmd” ile çıkılır.
C:\users\name\python
ya da
C:\users\name\py
Yazılarak python komut satırı açılır. Komut satırı artık “>>>” şeklinde görünecektir. Basit bir örnek verilirse;
>>> print(“merhaba dunya”)
Ekrana
Merhaba dunya yazacaktır.
Eğer blok program yazılacaksa bir text editorden yararlanmak gerekir, biz dersimizde “notepad++” kullanacağız.
Aynı komutu editörde yazıp, merhaba.py olarak kaydedelim ve işletim sistemi komut satırında
C:\users\name\python merhaba.py
Yazarak çalıştıralım.

Python Syntax (Yazım kuralları)

Pythonda bloklama için herhangi bir karakter kullanılmaz, girintilerle bloklar oluşturulur.

```
Diğer dillerin çoğunda şöyle yazılır;
İf(a>2)
{
Print("a degeri 2 den büyüktür")
}
Print("Bitti")
```

```
Pythonda ;
İf a>2:
  Print("a degeri 2 den büyüktür")
Print("Bitti")
```

Ekrana yazdırma

Sabit ifade ya da değişken içeriğini ekrana yazdırmak için print() komutundan yararlanılır.

Örnek:

```
x=5
adi="yahya"
print(x)        #ekrana 5
print(adi)      #ekrana yahya
print(x,adi)    #ekrana 5 yahya
```

Örnek:

```
x="Ankara "
y="Üniversitesi "
z="Fen Fakültesi"
print(x+y+z)    #ekrana Ankara Üniversitesi Fen Fakültesi
```

Python da Değişkenler

Pythonda değişken tanımı ayrıca yapılmaz, değişkene bir değer atandığında tanımlanmış olur.

```
Diğer dillerin çoğunda şöyle yazılır;
int x=3
String adi="yahya"
```

```
Pythonda ;
x=3
adi="yahya"
```

Değişken isimlendirme;

- Değişken adı bir harf veya alt çizgi ile başlamalıdır (Önerilen harfle başlaması),
- Değişken adı bir sayı ile başlayamaz (Yanlış örnek: 1sayi),
- Değişken adında harfler (Az), rakamlar(09) ve _ (alt çizgi) içermelidir,
- Değişken adları büyük/küçük harfe duyarlıdır (adi, ADI, Adi üç farklı değişkendir)
- Değişken adlarında TR karakter, boşluk, özel karakterler kullanılmaz (adı hatalı, dogrusu adi olmalıdır)
- Değişken adları python komutlarından herhangi birisi olamaz (Örnekler not, val, str, print…)

Değiken adları bir çok kelimeden oluşacaksa, daha okunaklı ve hemde dikkati çekmesi açısından kelimelerin ilk harfleri büyük yazılması önerilir.

Örnek ogrenci numarası için;

ogrenciNo, ogrNo, ogrNumara, ogrenciNumara gibi yazılmalıdır.

Değişkene Değer Atama

x=”Elma”
y=”armut”
z=”kiraz”
şeklinde her satırda yazılabileceği gibi tek satırda da yazılabilir.

Örnek:

x,y,z=”elma”,”armut”,”kiraz”
Aynı değer birden fazla değişkene de atanabilir.

Örnek:

x=y=z=100
Bir liste elemanlarını da değişkenlere aktarılabilir.

Örnek:

meyveler=[“elma”,”armut”,”kiraz”]
x,y,z=meyveler

Bellek Sızıntısı (Memory Leak)

Bir uygulamanın bellek harcarken kullandığı belleği işi bitince işletim sistemine geri veremediği duruma denilir. İş bitince serbest bırakılması gerekirken hala tutuluyor olması, RAM deki bellek miktarını aşırı şişirecek ve sistemin çökmesine sebep olacaktır. Bellek sızıntısı tüm dillerde bir sıkıntıdır.
Ancak Python da değişkenler çok fazla yer kapladığından bellek ciddi bir sorun olacaktır. Python’un sağladığı işlevsellik diğer dillere göre üstünlük olmasına karşın bellek tüketimi çok yüksektir. Bellek sıkıntısının olduğu bir yerde python bir dezavantaj olacaktır.

Pythonda Veri Türleri

Veri türü, bir değişkenin üzerindeki kısıtlamadır. Örneğin veri türü int olan olan değişken bir tam sayı içerir, veri türü float (kayan nokta) olan bir değişken ondalık değer içerir. Her değişkenin bir veri türü vardır. Diğer dillerde değişken tanımlanırken belirtmek zorundayız, ancak python da belirtmek zorunda değiliz, python bir değişkenin veri tipini kendisi yorumlar.

Sayısal : int, float, complex
String : str
Sıralı : list <<-[], tuple<<-() no element, range()
Sözlük : dict<<-{‘key’:’value’,…}
Set :set<<- {} no element
Boolean→True,False

Sayısal (Numerik)

```
x=5      #int
y=10.2 #float
```

String

```
adi=”yahya”            #string
soyadi=’demircan’      #string
```

String bilgi elemanlarına erişim

```
adi="nefise"
print(adi)             #nefise
print(adi[2])          #f      baştan 2. Elaman (ilk eleman indisi 0 dır !!!)
print(adi[-1])         #e      son eleman
```

```
print(adi[-3])          #i      sondan 3. eleman
print(adi[0:3])         #nef    [start:stop]
print(adi[3:])          #ise
print(len(adi))         #6
```

Sıralı (sequence)

Listeler: sıralı bir veri koleksyonudur. Diğer dillerdeki diziler gibidir. Burada elamanların aynı tür olması gerekmediğinden oldukça esnektir.

```
meyveler=["elma","armut","üzüm"]    #aynı tür bilgi
karisik=["ızgara","salata",300]     #farkli tur bilgi
hobiler=[]                          #boş liste
```

Liste elemanlarına erişim

```
karisik=["ızgara","salata",300]   #farkli tur bilgi
print(karisik[1])                 #string sonuc          salata
print(karisik[1:2])               #sonuc yine liste      [‘salata’]
print(karisik[-1])                # son eleman           300
```

Listeye Eleman ekleme

```
karisik.append("tatlı")
print(karisik)                    # ['ızgara', 'salata', 300, 'tatlı']
```

Liste Boyutunu alma

```
print(len(karisik))               #4
```

Listeden eleman Silme

```
karisik.remove(“tatlı”)           #değer vererek silme
print(karisik)                    # ['ızgara', 'salata', 300]
karisik.pop(1)                    #indis vererek silme
print(karisik)                    # ['ızgara', 300]
```

tuple: Listeye çok benzeyen nesne koleksiyonudur. Depolanan değerler farklı türde olabilir. Listeden ne farkı var sorusu akla gelebilir. Tuple’a yeni eleman eklenmez, bir eleman silinmez, silinecekse tümden silinir. Neden tuple sorusuna; değerleri değişmeyen yani sabit olan veri ve bellekte daha az yer kaplaması tercih sebebi olabilir

```
gunler=("p.tesi","salı","çarş","perş","cuma","c.tesi","pazar")
print(gunler)                     # ('p.tesi', 'salı', 'çarş', 'perş', 'cuma', 'c.tesi', 'pazar')
print(gunler[0])                  # ‘p.tesi’
print(gunler[-1])                 # ‘pazar’
print(gunler[0:3])                # ('p.tesi', 'salı', 'çarş')
print(len(gunler))                #7 elemanlı
```

range: Belirli aralıkta tam sayıları elde etmek için kullanılır. Genellikle for döngülerinde kullanılır.

```
sayilar=range(0,10)               #0 dahil 10 a kadar tam sayılar 0,1,..,9
print(sayilar)
ciftSayi=range(0,15,2)            #0 dahil 15 e kadar 2 artimla tam sayilar 0,2,4,…,14
print(ciftSayi[1])                #2
print(len(ciftSayi))              #8 elemanlı
```

sözlük (Dictionary): Bir anahtar üzerinden veri tutan ve aramayı bu anahtar üzerinden kısa sürede yapmaya yarayan veri türüdür. Anahtar ve değer çiftlerinden oluşur. Sıralanamaz ancak değiştirlebilir. Json veri formatına benzer, json formatındaki verilerin python dict yapısına aktarılarak işlenmesi sağlanır.

```
ogrenci={'adi':'yahya','soyadi':'demircan','sinifi':4}          #dict →{'key':value,…}
print(ogrenci['adi'])          #key'i adi olanın değeri yahya
ogrenci['adi']='Nefise'        #key'i adi olanı Nefise ile değiştirir
ogrenci['mobil']=555123456     #yeni item (öğe) ekler
print(ogrenci)                 #{'adi': 'Nefise', 'soyadi': 'demircan', 'sinifi': 4, 'mobil': 555123456}
del ogrenci['mobil']           #key'i mobil olan itemi siler
print(ogrenci)                 #{'adi': 'Nefise', 'soyadi': 'demircan', 'sinifi': 4}
print(len(ogrenci))            #3 item olduğunu verir
print('adi' in ogrenci)        # ogrenci içinde adi key'ini sorgular True
print('demircan' in ogrenci)   # deger sorgulamıyor False
```

Operatörler

Aritmetik Operatörler

```
a = 7
b = 3
print ('Toplam: ', a + b)          # Toplam: 10
print ('Fark: ', a - b)            # Fark: 4
print ('Çarpım: ', a * b)          # Çarpım: 21
print ('Bölüm: ', a / b)           # Bölüm: 2.333333
print (' Kat bölümü: ', a // b)          # Kat bölümü: 2   #kez var
print (Bölümden kalan: ', a % b)         # Bölümden kalan: 1    #modülo
print (üst alma: ', a ** b)              # üst alma: 343        #power
```

İlişkisel (Karşılaştırma) operatörleri

operatör	Tanım	Sözdizimi
>	Büyüktür: Sol işlenen sağdan büyükse doğrudur	x > y
<	Küçüktür: Sol işlenen sağdan küçükse doğrudur	x <y
==	Eşittir: Her iki işlenen de eşitse doğrudur	x == y
!=	Eşit değil – İşlenenler eşit değilse doğrudur	x != y
>=	Büyüktür veya eşittir: Sol işlenen sağdan büyük veya ona eşitse doğrudur	x >= y
<=	Küçüktür veya eşittir: Sol işlenen sağdan küçük veya ona eşitse doğrudur	x <= y

Koşul: İki değişken yada bilginin birbiriyle karşılaştırılmasıdır. Karışlatırmada ilişkisel operatörler kullanılır.

Örnek:

```
a=7
b=2
print(a>b)      #a>b koşulunun sonucu:True
print(a!=b)     #True
print(a==b)     #False
```

Mantıksal operatörler

İki koşulun sonuçlarına göre yeni bir karar oluşturmak için and, or ve not operatörleri kullanılır.
and: her iki koşulunda True olması, sonucunda True, aksi halde False olmasıdır
or:Koşullardan birisinin True olması, sonucunda True olması, akdi halde False olmasıdır.
not: Koşulun sonucunu tersine çeviririr.

Örnek:

```
a=7
b=3
c=0
a>b and c<10
```

birinci koşul (a>b) ile ikinci koşulun(c<10) birleştirilmesidir. Birinci koşulun sonucuna X, ikinci koşulun sonucuna Y denilirse (X ve Y nin sonuçları True yada False olacaktır), bu na göre aşağıdaki tablo oluşturulabilir.

X	Y	X and Y	X or Y	not(X)	not(Y)
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

```
a>b and c<10    >>>T and T    >>>Her ikisi de T olduğu için sonuc True olacaktır
a>0 or b>0      >>>T or T     >>> en az bir tane T olduğu için  sonuc True olacaktır
a>0 or b>10     >>>T or F     >>> en az bir tane T olduğu için sonuc True
a>0 and b>10    >>>T and F    >>>her ikiside T olmalıydı sonuc False
```

Atama Operatörleri

operator	Tanım	Sözdizimi
=	İfadenin sağ tarafındaki değeri sol taraftaki işlenene atayın	x = y + z
+=	Ekle ve Ata: Sağ taraftaki işleneni sol taraftaki işlenenle ekleyin ve ardından sol işlenene atayın	a += b a=a+b

operator	Tanım	Sözdizimi
-=	Çıkart VE: Sağ işleneni sol işlenenden çıkar ve ardından sol işlenene ata: Her iki işlenen eşitse doğru	a -= b a=a-b
*=	Çarp VE: Sağ işleneni sol işlenenle çarp ve ardından sol işlenene ata	a *= b a=a*b
/=	Böl VE: Sol işleneni sağ işlenenle bölün ve ardından sol işlenene atayın	a /= b a=a/b
%=	Katsayı VE: Sol ve sağ işlenenleri kullanarak modülü alır ve sonucu sol işlenene atar	a %= b a=a%b

Örnek:

```
a=4             #atama
b=5
a+=b            #a=a+b
print(a)        #9
a/=b            #a=a/b
print(a)        #1.8
a=10
a%=3            #a=a%3
print(a)        #1
x=y             # error çünkü y değişkeni ortamda yok
x+=y            # error
```

+ operatörü string verileri birleştirecektir.

Örnek:

```
k=3
y=”yahya”
print(k+y)      #hataya sebep olacaktır. Virgül kullanılarak yazdırlabilir
print(k,y)      #3 yahya
```

Tür Dönüşümü:

Örtülü dönüşüm: Python yorumlayıcısı otomatik olarak veri dönüşümü yapar.

Örnek:

```
x=5
y=10.2
z=x+y    #z otomatik olarak float oldur. Veri kaybı olmaması için int değil float dönüştü
```

Açık Dönüşüm: Kullanıcı gereksinimine göre manuel olarak değiştirilir, veri kaybı riski olabalir.

int(a,base)

```
s="101"         #string
print(int(s))   #taban yazılmaz ise 10 tabaninda
print(int(s,2))
```

float(s)

```
s="101"
print(float(s))   #101.0
```

ord() ve chr()

```
s="A"
print(ord(s))   #simgesi A olanın aski kodu 65 verir
x=65
print(chr(x))   #Asci kodu 65 olan sayının karakterini verir
```

str()

```
y=15.5
print(str(y))
print("*"+str(y)+"*")     #üç str birleştirilebilir oldu
```

tuple() : Bu fonksiyon bir demete dönüştürmek için kullanılır .

set() : Bu işlev, set'e dönüştürüldükten sonra türü döndürür .

list() : Bu fonksiyon , herhangi bir veri tipini bir liste tipine dönüştürmek için kullanılır .

Örnek:

```
metin="Ankara"
t=tuple(metin)
print(t)            #('A', 'n', 'k', 'a', 'r', 'a')
print(t[1])         #n

s=set(metin)
print(s)            #{'a', 'r', 'A', 'n', 'k'}
#print(s[1]) #set türünde erişim yoktu, for ile erişilebilir
for i in s:
   print(i)

listem=list(metin)
print(listem)               #['A', 'n', 'k', 'a', 'r', 'a']
print(listem[1])            #n
```

```
a       #rasgele görüntü, yinelenen
r       harf yok eder!
A
n
k
```

Pythonda Girdi (input)

Programcılıkta yazılan program kullanıcılardan veri alma ihtiyacı duyabilir. Bu veri alma yollarından birisi de klavyedir. Çoğu programda olduğu gibi pythonda da input kullanılır.

degisken=input(“mesaj”)

Örnek:

Klavyeden isim girip ekrana yazdıralım.

```
adi=input("isminizi giriniz:")
print(adi)
```

Örnek:

İki sayı giriniz, toplamını ekrana yazdırınız.

```
x=input("ilk sayıyı giriniz:")              #klv girilen her zaman string unutmayalım
y=input("ikinci sayıyı giriniz:")
z=float(x)+float(y)
print(z)
```

Pythonda Kontrol Akışı-Karşılaştırma

İf ifadesi: En basit karar verme ifadesidir. Belirli ifade yada ifade bloğunun yapılıp yapılmamasına karar vermek için kullanılır.

```
if koşul:
  #koşulun sonucu True ise yapılacaklar
```

Pythonda blok ifadesi girintilerle yapılır. İf bloğunun bittiğini, if ile aynı hizadaki ifade belirler.

Örnek:

```
x=10
if x>5:
  print(“x 5 den büyüktür”)
  print(“if in içi”)
print(“if in bittiği yerdir, her zaman calisir”)
```

if-else İfadesi

```
if koşul:
  #koşul doğru ise yapılacaklar
else:
  #koşul doğru değilse yapılacaklar
```

Örnek:

```
x=10
if x>5:
   print(“x 5 den büyüktür”)
   print(“true blok ici”)
else:
   print(“x 5 den büyük değildir”)
   print(“else blok ici”)
a=3    #if blogunun bittiği satır, her zaman çalışır
print(“program bitti”)
```

Örnek:

Klavyeden girilen sayı rakam mıdır ? Ne olursa olsun 2 katını da yazdırınız.

```
sayi=int(input("sayı giriniz"))              #sayı girilir girilmez int çevirilip atanıyor
if sayi>0 and sayi<10:
   print("girilen ",sayi," sayısı rakamdır")
else:
   print("girilen ",sayi," sayısı rakam değildir")
print(sayi*2)
```

if-elif-else ifadesi:

Karar verilecek ifade bazen birden fazla duruma sahip olabilir. Bu durumlara karşı yapılacaklar için karar vermede if-elif kullanmak daha hızlı ve anlaşılır olur.

```
if kosul1:
  #koşul1 true ise yapılacaklar
elif koşul2:
 #koşul2 true ise yapılacaklar
…
else:
  #yukarıdaki koşullara uymadığında yapılacak
```

Örnek:
Klavyeden girilen sayının 100 sayısına göre durumu nedir? Her durumda sayinin 100 den farkını yazdırınız.
Sayı 100 den büyük olabilir, küçük olabilir ya da eşittir.

```
sayi=int(input("sayı giriniz"))
if sayi>100 :
   print("girilen ",sayi," sayısı 100 den büyük")
elif sayi<100:
   print("girilen ",sayi," sayısı 100 den küçük")
else:
   print("girilen ",sayi," sayısı 100 e eşit")
print(sayi-100)
```

Tekrarlama Yapıları – Döngüler
Bir kod blogunun birden fazla kez tekrarlanması, ardışık sayılara göre işlem yapılması, bir dizinin elemanlarına erişim yada dosya okunması gibi durumlarda kullanılır.

For döngüleri
Pythonda diğer dillerde olduğu gibi for(i=0;i<n;i++) gibi bir kalıp yoktur. Pythonda sıralı geçiş yani string, list, tuple, set, dict yada range gibi bir yineleme yapmak için kullanılır.

For döngüsü ve String:
String ifadenin her bir elemanı ele almak için kullanılabilir.

Örnek:

```
metin="Ankara"
for harf in metin:
   print(harf)
```

```
A
n
k
a
r
a
```

metin değişkenin her bir elemanı harf değişkenine aktarır ve metin değişkenin sonuna kadar devam eder.

Örnek:
Klavyeden girilen cümlede a harfinden kaç tane vardır. Bunu yapacak fonksiyon var ama biz konuyu anlayabilmek için kendimiz kod yazalım.

```
metin=input("Bir cumle giriniz")
adet=0
for i in metin:
   if(i=='a'):
      adet+=1
print(adet," tane a vardır")
```

For döngüsü ve List:
Listenin her bir elemanına erişmek için kullanılır. Her bir eleman i değişkenine liste sonu gelinceye kadar atar.

Örnek:

```
meyveler=["elma","armut","kiraz","kayısı"]
for i in meyveler:
  print(i)
```

For döngüsü ve Range:

Ardışık sayılarla işlem yapmak için range ile beraber kullanılır.
Range(start,stop[,step]) bu fonksiyon start dahil, stop değerine kadar (hariç) verilen step kadar artırarak tamsayılar üretir. Step yazılmaz ise varsayılan artım 1 dir.

Örnek:

1 den 10 kadar olan tam sayıları ekrana yazdıralım.

```
for i in range(1,10):
  print(i)                  # 1 2 3 4 5 6 7 8 9 alt alta yazdırılır.
```

Örnek:

Çift rakamları ekrana yazdıralım.

```
for i in range(0,10,2):
  print(i)                  #0 2 4 6 8 alt alta yazdırılır.
```

Örnek:

Klavyeden girilen sayının rakamları toplamını bulalım.

```
sayi=input("Bir sayı giriniz")
toplam=0
for i in sayi:              #klv girilen bilgi string idi, böylece string üzerinde ilerleme yapılır.
   toplam+=int(i)           #i ler string unutulmamalı
print(toplam)
```

while Döngüsü

Belirtilen bir koşul doğru olduğu sürece while bloğu içerisindeki kodlar tekrar edilir.

```
While koşul:
  Yapılacaklar
```

Örnek:

Rakamları ekrana yazdıralım.

```
i = 0
while i < 10:
  print(i)
  i += 1
```

Akla şu soru gelebilir, bu işlemi for ile daha kolay yapabilirdik. Bu soru için doğru olur.

Örnek:

0 ile 2pi arasındaki değerlerini 0.1 artımlarla ekrana yazdıralım. Burada artım miktarı ondalık olduğundan for burada kullanılmaz.

```
i=0
while i<2*3.1415:
   print(i)
   i+=0.1
```

Soru:
y=1,2,3… değerleri için x=x+y değerini x>30 oluncaya kadar hesaplayarak x değerini ekrana yazdırınız.

Soru:
1+2+2+3+3+3+4+4+4+4+… bu örüntünün sonucunu 1920 ye kadar hesaplayıp ekrana yazdırınız.

İmport-Modül Çağırma
Daha önceden bizim yazdığımız yada başkalarının yazdığı python programını (modülünü) çağırma işlemidir.
Örneğin belirli aralıkta rasgele tam sayıya ihtiyacımız var. Bunun için yazılmış hazır modüller vardır ondan yararlanmak için import edilmesi gereklidir.

Örnek:
```
import random
print(random.randint(1, 16))     #1 ile 16 arasında rasgele tam sayı üretir.
```

Örnek:
Bilgisayarın zamanına ihtiyacımız varsa datetime modülü import edilir.
```
import datetime
current_time = datetime.datetime.now()
print(current_time)                          # 2023-08-08 13:57:23.186969
```

Benzer şekilde şunlarda elde edilebilir.
```
print("Year :", current_time.year)
print("Month : ", current_time.month)
print("Day : ", current_time.day)
print("Hour : ", current_time.hour)
print("Minute : ", current_time.minute)
print("Second :", current_time.second)
print("Microsecond :", current_time.microsecond)
```

Zamanı Biçimlendirme için time modülü kullanılabilir.
Örnek:
```
import time
curr_time = time.strftime("%H:%M:%S", time.localtime())      # Tarih için "%d:%m:%Y"
print("Şu anki saat  ", curr_time)                           # Şu anki saat 14:01:06
```

Yapmak istediğiniz programa göre daha önce hazırlanmış bir çok modul bulabilirsiniz.
```
Pyhon girdi/çıktı, path listesi, modül listesi      : import sys
İşletim sistemi komutlarını yürütmek için      : import os      #liste=os.listdir(path)
Matematik işlemler için                        : import math
```

Kütüphane Kurulumu
Bazı moduller python ile yerleşik gelmeyebilir. Harici modül kullanmak için gerekli kütüphaneyi install (yükleme) gereklidir, sonra import edilir.

Örneğin Numpy kütüphanesi dizilerle çalışmak üzere kullanılan bir kütüphanedir. Pyhon liste' lerine göre çok hızlı çalışmakta ve işlevselliği fazladır. Numpy kullanmak için öncelikle install edilmesi gereklidir.

Pip sisteminizde kurulu ise (pip kurulumu : `c:\>python -m pip install -U pip`)
C:\>pip install numpy
Artık numpy kütüphanesi çalışmanıza dahil edilebilir.

Örnek:

```
import numpy
dizi = numpy.array([1, 2, 3, 4, 5])
print(dizi)
print(numpy.sum(dizi))
```

numpy fonksiyonlarına erişirken **numpy.sum** şeklinde yazılmaktadır. Burada numpy kelimesi yerine kısaltma olarak np kullanılmak istenirse yukarıdaki program aşağıdaki gibi yazılmalıdır.

```
import numpy as np
dizi = np.array([1, 2, 3, 4, 5])
print(np.sum(dizi))
```

Normalde python listesinde dizinin elemanlarının toplamını direk bulamazken np de bu mevcut. Sayılarla uğraşan yazılımcılar daha çok np'yi tercih ederler.

Örnek:
Şu şekilde bir listemiz olsun sayilar=[5,7,3,2,15,3,6,8]
Biz bu sayıların ortalamasından küçük olanların listesini görmek istersek bunu normal koşullarda standart (soldaki) kodlarla yazarsak : ort:6.125

```
sayilar=[5,7,3,2,15,3,6,8]
toplam=0
for sayi in sayilar:
   toplam=toplam + sayi
ort=toplam/len(sayilar)
print(ort)
for sayi in sayilar:
   if(sayi<ort):
      print(sayi)   #5 3 2 3 6
```

```
import numpy as np
dizi=np.array([5,7,3,2,15,3,6,8])
print(dizi[dizi<np.average(dizi)])          # [5 3 2 3 6]
print(np.where(dizi<np.average(dizi)))   #(array([0, 2, 3, 5, 6], dtype=int64),)
```

np ile daha az ifadelerle aynı sonuca erişilebilir. Np.where ifadesiyle de bu sayıların konumunu elde edebilirsiniz.

Soru:
16,7,13,12,15,33,26,18,14 sayılarının median değerini bulunuz. A) Numpy kullanmadan, b) numpy kullanarak
Median :sıralanmış verilerin ortasındaki değeridir. Dizimiz 9 elemanlı, ortası 5 dir

[7, 12, 13, 14, 15, 16, 18, 26, 33]

```
import math
dizi=[16,7,13,12,15,33,26,18,14]
dizi.sort()
orta=math.floor(len(dizi)/2)
print(dizi[orta])
```

```
import numpy as np
dizi=[16,7,13,12,15,33,26,18,14]
print(np.median(dizi))
```

dizi çift sayıda olursa nasıl olacak

Kullanıcı Fonksiyonları:
Bir kod blogu birden fazla kez yazılacaksa bu yapılmaz, yerine bu kod bloğu fonksiyon haline getirilir. Kaç kez istenirse çağırılır. Dahası bu fonksiyon başka programlarda da çağırılarak kullanılır. Bir kez yaz, defalarca kullan. Böylelikle zamandan kazanılır, kod karmaşasından kurtulunur ve müdahale gerektiğinde tek yerden yapılarak hataların önüne geçilir.

Buna örnek genelde kombinasyon sorusuna cevap aranılarak verilir. C(8,5)=8!/((8-5)!*5!
Bu bağıntıda üç kez faktöriyel almak gerekiyor, normal programlama mantığıyla şöyle (solda)

```
x=1  #8! için
y=1  #(8-5)! için
z=1  #5! için
for i in range(1,8+1):
   x=x*i
for i in range(1,8-5+1):
   y=y*i
for i in range(1,5+1):
   z=z*i

print(x/(y*z))            #56.0
```

```
def faktor(a):
 f=1
 for i in range(1,a+1):
   f=f*i
 return f

print(faktor(8)/(faktor(8-5)*faktor(5)))        #56.0
```

Bu faktoriyel içerek modülümüzü matematik.py olarak kaydedip başka bir programda kullanabiliriz. Daha önce import yapmayı görmüştük. Şimdi başka bir örnekte matematik.py kullanacağız. Matematik.py ile ornek uygulama aynı dizinde olmalıdır (Şimdilik).

Örnek:
```
import matematik
print(matematik.faktor(5))
```

çıktıda 56.0 ve 120 veriyor, bunun sebebi en alttaki print satırını silmek olmalıdır (Şimdilik). Ayrıca kullanırken kısaltma kullanılırsa örneğimiz şu şekilde olmalıdır.
```
import matematik as mat
print(mat.faktor(5))
```

from – import ile kısmi dahil etme
Bir kütüphaneyi import ile dahil ederken o kütüphanedeki tüm modüllerin yerine bize gereken modülü dahil edebiliriz.

Örnek:
math kütüphanesinde onlarca modül sin, cos, …, exp, log, factorial,…,pi vardır. Bize sadece pi sayısı gerekli diğerleri kullanmayacağız o zaman programımız şu şekilde olmalıdır.

```
from math import pi
print(pi)
```

Dosya İşlemleri
Verilerin kalıcı olarak HDD saklanması önemlidir. Veriler HDD de dosya içerisinde saklanır. Dosya işlemleri olarak adlandırlan bu işlemde yazma yada okuma olarak işlem yapılır.

Dosyalarda gelişigüzel veri de saklanabilir, ya da belirli bir formatta (düzende) veriler de saklanabilir.

Örnek: hatirlat.txt
Selam dünya
Ay sonlarında borçlar ödenecek
Haziran ayında final sınavı var
…
Bu verilerin bir düzeni yok, kişisel verileri hatırlamak amacıyla dosyada saklanır.

Örnek: Biraz düzenli veriler için rehber.txt
Yahya Demircan 05351234567
Emrullah Ecik 0542987456
Muhittin 05331234567
Sabri Kerem 05221235456

Bu dosya da bize biraz da olsun düzenli veri tutuluyor izlenimi veriyor. Peki bu dosyayı okurken birilerinin bilgileri tam iken, birilerinin bilgisi eksik bu da bizi hataya götürür mü. İnsan gözü bunu görüyor, ama programcılıkta bu işler böyle olmuyor. Herkesin soyadı var mı ?, iki isimli olan var mı ?

Örnek: Düzenli veriler için rehber.txt şöyle olmalıdır.
Yahya;Demircan;05351234567
Emrullah;Ecik;0542987456
Muhittin; ;05331234567
Sabri Kerem; ;05221235456

Bu dosya 4 sütun bilgi içeriyor, kimisinin soyadı yok, kimisi iki isimli olarak yer alıyor. Okuma işleminde hataya sebep olmayacaktır. Burada verileri bir birinden ayıran bir işaret var. Buna ayraç (**delimiter**) denilir.
Ayraç; boşluk, virgül, noktalı virgül gibi herhangi bir karakter olabilir.
Görünmeyen işaretler: Hareketli dosya imleci, her satır sonunda EOL, dosyanın en altında EOF gibi işaretler bulunur. Bunlar bize dosya işlemede yardımcı olurlar.

Dosyalar üzerinde farklı işlemler yapılır. Buna veri işleme denilir ve Bunlar;

- Yeni veri ekleme,
- Verileri sorgulama,
- Veri Güncelleme,
- Kayıt Silme

işlemlerinden oluşur.

Yukarıdaki işlemlerin yapılabilmesi için;

- Dosya açılır
- Okuma/Yazma yapılır
- Dosya kapatılır.