

Pyhton Programlama

Giriş

DR. ŞAFAK KAYIKÇI



Python



Python'u büyük bir yılan olarak bilseniz de, Python programlama dilinin adı, Monty Python'un Flying Circus adlı eski bir BBC televizyon komedi skeç serisinden gelmektedir.



Guido
van
Rossum

Python, Hollandalı Guido van Rossum tarafından 1991'de geliştirildi.

Özellikleri

- Basit, kullanımı kolay sözdizimine sahiptir.
- Windows, Linux'un tüm dağıtımları, Mac OS X, Unix vb. Dahil hemen hemen her platformda çalışabilir.
- C, C ++ vb. gibi diğer programlama dillerinin parçaları python ile birleştirebilir.
- Geniş standart kitaplık desteği bulunur.
- Açık kaynak kodludur ve aktif olarak çalışan büyük bir topluluk desteğine sahiptir.
- Nesneye yönelik programlama dilidir.
- Kodu satır satır çalıştıran ve hata ayıklamayı kolaylaştıran yorumlana (Interpreted) bir dildir.
- CPython (Python'un Standart uygulaması), Jython (Java programlama diliyle entegrasyon için hedeflenmiştir) gibi çok sayıda uygulamaya sahiptir.



Uygulamaları

- Web Uygulaması: Python, ölçeklenebilir ve güvenli web uygulamaları geliştirmek için kullanılabilir. Django, Flask, Pyramid vb. web tabanlı uygulamaları tasarlamak ve geliştirmek için kullanılabilecek altyapılardır. (instagram, reddit, mozilla, Google vs altyapısında Python vardır)
 - Masaüstü Uygulamaları: GUI geliştirmek için Tk ve Kivy gibi araçlara sahiptir.
 - Bilimsel Hesaplamalar: SciPy ve NumPy gibi Python kitaplıkları bilimsel hesaplamalar için kullanıp, bu hesaplamalarda oldukça hızlıdır.
 - AI ve ML (Yapay Zeka ve Makine Öğrenimi): Python, Yapay Zeka ve Makine Öğrenmesinde en fazla kullanılmakta olan programlama dilidir.
 - Görüntü İşleme: Python, herhangi bir görüntünün piksel piksel geçmesini ve analiz edilmesini içeren Pillow, scikit-image gibi güçlü görüntü işleme kütüphanelerine sahiptir.
- ❌ Düşük seviyeli programlama ve mobil cihazlar için uygulamalarda Python başarılı değildir (henüz).

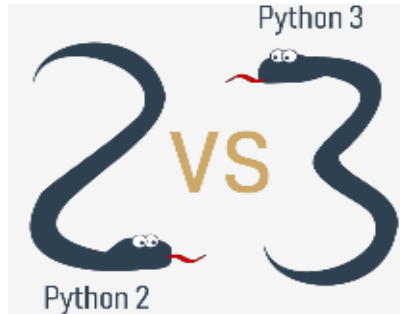


Pyhton 2 - Pyhton 3

Python 2, orijinal Python'un eski bir sürümüdür. Gelişimi o zamandan beri kasıtlı olarak durdurulmuştur ancak bu herhangi bir güncelleme olmadığı anlamına gelmiyor. Aksine, güncellemeler düzenli olarak yayınlanmaktadır, ancak dili önemli bir şekilde değiştirmeyi amaçlamazlar. Yeni keşfedilen hataları ve güvenlik açıklarını düzeltmeyi tercih ederler.

Python 3, dilin daha yeni (kesin olarak güncel) sürümüdür. Kendi gelişim yolunda ilerleyip kendi standartlarını yaratmaktadır.

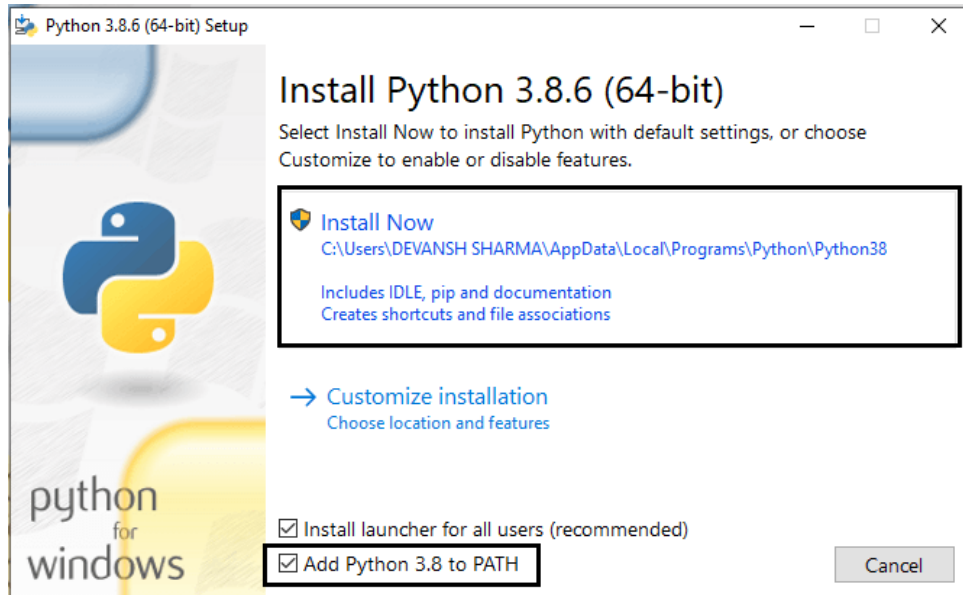
Python'un bu iki sürümü birbiriyle uyumlu değildir. Python 2 komut dosyaları bir Python 3 ortamında çalışmaz (tersi de geçerlidir).



Pyhton - Yükleme

Windows

python.org/download



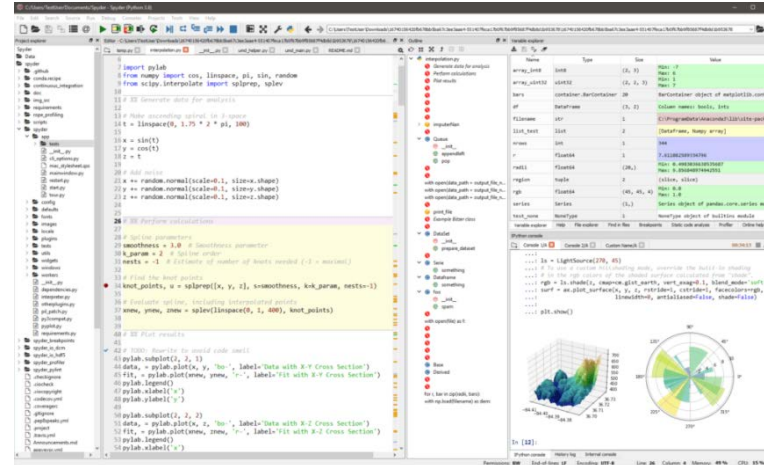
Linux ve Mac (Python 2) için pyhton sistemde yüklü olarak gelmektedir.

Pyhton 3 için yükleme gerekecektir.

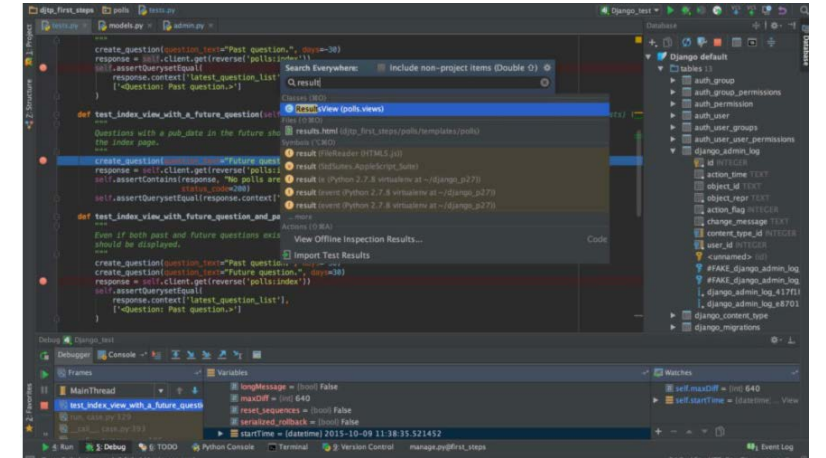
Geliştirme Ortamları (IDE)



IDLE (default)



Spyder



PyCHARM

Yazım Kuralları

- Büyük / küçük harfe duyarlıdır.
- Path yazımları için forward slash (/) kullanılır.

Windows : `C:\folderA\folderB` Python: `C:/folderA/folderB`

- Bir komut sonlandırıcı yoktur. Tek bir satıra iki ayrı çalıştırılabilir ifade yazmak için, komutları ayırmak için noktalı virgül (;) kullanılır.

```
print ("Hello, World!")
```

```
print ("Hello, World!") ; print ("This is second line")
```

- String ifadeler için tek tırnak ", çift tırnak ""ve hatta üçlü tırnak """"kullanılabilir.
- Yorum satırı için # kullanılır.

- **Kod Girintisi (Code Indentation)**: Bu, python programlamanın en önemli kuralıdır. Girinti için **tab** kullanmanız önerilir

X

```
if True:  
print ("Yes, I am in if block");
```

✓

```
if True:  
    # this is inside if block  
    print ("Yes, I am in if block")
```


Değişkenler

- Değişken türünü belirtilmesine gerek yoktur.
- Büyük / küçük harfe duyarlıdır.
- Değişkenin ilk karakteri bir alfabe veya alt çizgi (_) olmalıdır.
- İlk karakter dışındaki tüm karakterler, küçük harf (az), büyük harf (AZ), alt çizgi veya rakamdan (0-9) oluşabilir.
- Herhangi bir boşluk veya özel karakter (!, @, #, %, ^, &, *) içermemelidir.
- Python'da tanımlanan herhangi bir anahtar kelimeye olmamalıdır

Anahtar Kelimeler

| | | | | |
|----------|---------|-------|----------|--------|
| True | False | None | and | as |
| asset | def | class | continue | break |
| else | finally | elif | del | except |
| global | for | if | from | import |
| raise | try | or | return | pass |
| nonlocal | in | not | is | lambda |

Nesne Referansları

Python, yüksek düzeyde nesne yönelimli programlama dilidir; bu nedenle her veri ögesi belirli bir sınıfa aittir.

```
type("safak")
```

```
<class 'str'>
```

```
a = 50
```

```
b = a
```

```
print(id(a))
```

```
print(id(b))
```



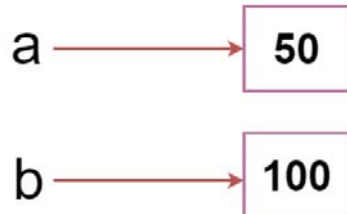
b değişkeni, Python başka bir nesne yaratmadığı için, a'nın işaret ettiği aynı nesneyi ifade eder.

id () fonksiyonu, nesne kimliğini tespit etmek üzere kullanılır

```
a = 50
```

```
b = 100
```

```
print(id(b))
```



Her iki değişken de farklı nesneleri gösterecektir. Aynı değişkeni iki farklı değere atarsak Python belleği verimli bir şekilde yönetir.

Değişken Adlandırma Yöntemleri

Camel Case - ortasındaki her kelime veya kısaltma büyük harfle başlar. Boşluğun etkisi yoktur. Örneğin : ogrenciAdi, kayitSirasi vb.

Pascal Case - Camel Case ile aynıdır, ancak burada ilk kelime de büyük harftir. Örneğin : OgrenciAdi, KayitSirasi vb.

Snake Case- Kelimeler alt çizgiyle ' _ ' ayrılır ve her kelime küçük harf ile başlar. Örneğin : öğrenci_adı, kayit_sirasi

Kebab Case - Kelimeler arası '-' (tire) ile ayrılır, ve her kelime küçük harf ile başlar. Örneğin : öğrenci-adi, kayit-sirasi

Hungarian Notation (Macarca Gösterimi) - Değişkenlerin önüne değişkenle ilgili bir kaç önek (prefix) eklenerek değişken hakkında bilgi verilmesi sağlanmıştır. Örneğin : sOğrenciAdi, iKayitSirasi

Upper Case - bütün kelimeler büyük harfle yazılır ve kelimelerin arası ' _ ' (alt çizgi) ile ayrılır. Örneğin : OGRENCI_ADİ, KAYIT_SIRASI

Operatörler

Operatörler değişkenler arasındaki işlemleri gösteren sembollerdir.

Python, aşağıdaki operatörleri destekler :

- Aritmetik operatörler (Arithmetic operators)
- Karşılaştırma operatörleri (Comparison operators)
- Atama Operatörleri (Assignment Operators)
- Mantıksal operatörler (Logical Operators)
- Bitsel Operatörler (Bitwise Operators)
- Üyelik Operatörleri (Membership Operators)
- Kimlik Operatörleri (Identity Operators)

Aritmetik operatörler

| Operatör | Açıklama |
|------------------|--|
| + (Toplama) | İki işlenen toplamak için kullanılır. Örneğin, $a = 20$, $b = 10 \Rightarrow a + b = 30$ |
| - (Çıkarma) | İkinci değişkeni ilk değişkenden çıkarmak için kullanılır. İlk değişken ikinci işlenenden küçükse, değer negatif sonuçlanır. Örneğin, $a = 20$, $b = 10 \Rightarrow a - b = 10$ |
| / (Bölme) | İlk değişkeni ikinci değişkene böldükten sonra bölümü döndürür. Örneğin, $a = 20$, $b = 10 \Rightarrow a / b = 2.0$ |
| * (Çarpma) | Bir değişkeni diğeriyle çarpmak için kullanılır. Örneğin, $a = 20$, $b = 10 \Rightarrow a * b = 200$ |
| % (mod) | İlk değişkeni ikinci değişkene böldükten sonra kalanı döndürür. Örneğin, $a = 20$, $b = 10 \Rightarrow a \% b = 0$ |
| ** (Üs) | İlk değişkenin ikinci değişken üssü olarak hesaplanır. Örneğin, $3^{**}2 \Rightarrow 9$ |
| // (taban bölme) | İki değişkeni bölerek bölümün taban değerini verir. Örneğin, $9//2 \Rightarrow 4$ |

Karşılaştırma operatörü

| Operatör | Açıklama |
|----------|------------|
| == | Eşit |
| != | Eşit değil |
| <= | Küçük eşit |
| >= | Büyük eşit |
| > | Büyük |
| < | Küçük |

Atama Operatörleri

| Operatör | Açıklama |
|----------|--|
| = | Sağ ifadenin değerini sol değişkene atar. |
| + = | Soldaki değişkenin değerini sağ değişken değeri kadar artırır ve değiştirilen değeri tekrar sol değişkenin atar. Örneğin, $a = 10$ ise, $b = 20 \Rightarrow a + = b$, $a = a + b$ 'ye eşit olacak ve bu nedenle, $a = 30$ olacaktır. |
| - = | Sol değişkenin değerini sağ değişken değeri kadar azaltır ve değiştirilen değeri tekrar sol değişkene atar. Örneğin, $a = 20$ ise, $b = 10 \Rightarrow a - = b$, $a = a - b$ 'ye ve dolayısıyla $a = 10$ 'a eşit olacaktır. |
| * = | Soldaki değişkenin değerini sağ değişkenin değeriyle çarpar ve değiştirilen değeri sonra sol değişkene geri atar. Örneğin, $a = 10$ ise, $b = 20 \Rightarrow a * = b$, $a = a * b$ 'ye ve dolayısıyla $a = 200$ 'e eşit olacaktır. |
| % = | Soldaki değişkenin değerini sağ değişkenin değerine böler ve kalanı tekrar sol işlenene atar. Örneğin, $a = 20$ ise, $b = 10 \Rightarrow a \% = b$, $a = a \% b$ 'ye eşit olacak ve bu nedenle, $a = 0$ olacaktır. |
| ** = | $a ** = b$, $a = a ** b$ 'ye eşit olacaktır, örneğin, eğer $a = 4$, $b = 2$, $a ** = b$, a 'ya $4 ** 2 = 16$ atayacaktır. |
| // = | $a // = b$, $a = a // b$ 'ye eşit olacaktır, örneğin, eğer $a = 4$, $b = 3$ ise, $a // = b$, a 'ya $4 // 3 = 1$ atayacaktır. |

Bitsel Operatörler

| Operatör | Açıklama |
|--------------------|---|
| & (ve) | İki değişkende aynı yerdeki her iki bit 1 ise, sonuca 1 kopyalanır. Aksi takdirde 0 kopyalanır. |
| (veya) | Her iki bit de sıfırsa ortaya çıkan bit 0 olacaktır; aksi takdirde ortaya çıkan bit 1 olacaktır. |
| ^ (xor) | Her iki bit de farklıysa ortaya çıkan bit 1 olacaktır; aksi takdirde ortaya çıkan bit 0 olacaktır. |
| ~ (ters) | Değişkenin her bitinin tersini hesaplar, yani bit 0 ise, ortaya çıkan bit 1 olacaktır ve bunun tersi de geçerlidir. |
| << (sola kaydırma) | Sol değişken değerini, sağ değişkende bulunan bit sayısı kadar sola hareket ettirilir. |
| >> (sağa kaydırma) | Sol değişken, sağ değişkende bulunan bit sayısı kadar sağa hareket ettirilir. |

Mantıksal operatörler

| Operatör | Açıklama |
|----------|---|
| and | Her iki ifade de doğruysa, koşul doğru olacaktır. Eğer a ve b iki ifade ise, $a \rightarrow \text{doğru}$, $b \rightarrow \text{doğru} \Rightarrow a \text{ ve } b \rightarrow \text{doğru}$. |
| or | İfadelerden biri doğruysa, koşul doğru olacaktır. A ve b iki ifade ise, $a \rightarrow \text{doğru}$, $b \rightarrow \text{yanlış} \Rightarrow a \text{ veya } b \rightarrow \text{doğru}$. |
| not | Bir ifade a doğruysa, o zaman değil (a) yanlış olur ve bunun tersi de geçerlidir. |

Üyelik Operatörleri

| Operatör | Açıklama |
|----------|---|
| in | İlk değişken ikinci değişken (liste, tuple veya sözlük) içerisinde <u>bulunursa</u> doğru olarak değerlendirilir. |
| not in | İlk değişken ikinci değişken (liste, tuple veya sözlük) içerisinde <u>bulunmazsa</u> doğru olarak değerlendirilir. |

Kimlik Operatörleri

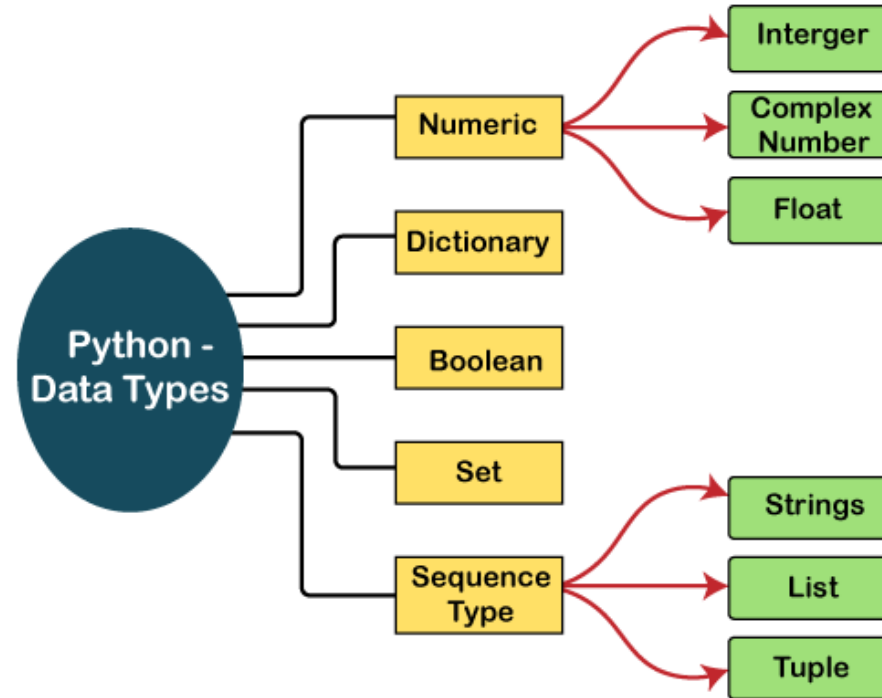
| Operatör | Açıklama |
|----------|--|
| is | Her iki tarafta bulunan referans aynı nesneyi gösteriyorsa doğru olarak değerlendirilir. |
| is not | Her iki tarafta bulunan referans aynı nesneye işaret etmiyorsa doğru olarak değerlendirilir. |

Python Programlama Veri Tipleri

DR. ŞAFAK KAYIKÇI

Veri Türleri

Python'da değişkeni tanımlarken değişkenin türünü tanımlamamıza gerek yoktur.



Sayılar

- 1.Int** - Tamsayı değeri, 10, 2, 29, -20, -150 gibi tam sayılar gibi herhangi bir uzunlukta olabilir. Python'da bir tamsayı uzunluğu üzerinde herhangi bir sınırlama yoktur. Değeri int'e aittir
- 2.Float** - Float, 1.9, 9.902, 15.2, vb. gibi noktalı sayıları saklamak için kullanılır. 15 ondalık basamağa kadar kesinliği vardır.
- 3.Complex**- Karmaşık bir sayı sıralı bir çift içerir ($x + iy$) burada x ve y sırasıyla gerçek ve sanal kısımları gösterir. Örn : 2.14j, 2.0 + 2.3j

Sıralı Tipler - Listeler

Listeler, çeşitli veri türlerinin sıralı olarak saklamak için kullanılır. Python listeleri değiştirilebilir (mutable) türdendir, yani oluşturduktan sonra elemanını değiştirebilir.

Listede saklanan öğeler virgülle (,) ayrılır ve köşeli parantezler [] içine alınır.

Listenin elemanların indeks ile erişebilir.

```
list1 = [1, 2, "Safak", 3]
list2 = [1, 2, 3, "Safak"]
print(list1 == list2 )
print(list1[0])
print("id:%d isim:%s"%(list1[0], list1[2]))
```


Liste indeksleme ve bölme

Listenin elemanlarına dilim operatörü [] kullanılarak erişilebilir. İndeks 0'dan başlar ve sonuncusu (length – 1)'dir.

```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
```

```
print(list[0:6])
```

```
print(list[:])
print(list[2:5])
print(list[1:6:2])
```

```
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

list_variable(start:stop:step)

List = [0, 1, 2, 3, 4, 5]

Forward Direction



0

1

2

3

4

5

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|

-6

-5

-4

-3

-2

-1



Backward Direction

Liste – Eleman değiştirme, ekleme, silme

```
list = [1,2,3,4,5,6,7]
```

```
list[2] = 10
```

```
print(list)
```

```
list[1:3] = [89, 78]
```

```
print(list)
```

```
# listenin sonuna ekler
```

```
list[-1] = 25
```

```
print(list)
```

```
#ikinci elemanı siler
```

```
del(list[2])
```

```
print(list)
```

Python ayrıca listeye değer eklemek için kullanılabilen **append** ve **insert** yöntemleri sağlar.

del sözcüğü kullanılarak silme yapılabilir . Python , listeden hangi öğenin silineceğini bilmiyorsa , bize **remove** yöntemini de sağlar

Liste İşlemleri

L1 = [1 , 2 , 3 , 4] ve L2 = [5 , 6 , 7 , 8]

| | | |
|-----------------|---|--|
| Tekrarlama (*) | Tekrar operatörü, liste öğelerinin birçok kez tekrarlanmasını sağlar. | L1 * 2 = [1, 2, 3, 4, 1, 2, 3, 4] |
| Birleştirme (+) | Operatörün her iki tarafında belirtilen listeyi birleştirir. | L1 + L2 = [1, 2, 3, 4, 5, 6, 7, 8] |
| Üyelik | öge listede varsa true, yoksa false döndürür. | print(2 in L1) → True |
| Tekrar | For döngüsü, liste öğelerini yinelemek için kullanılır. | for i in L1: print(i) Output 1 2 3 4 |
| Uzunluk | Listenin uzunluğunu almak için kullanılır | len (L1) = 4 |
| max | Listenin maksimum ögesini döndürür. | max(L1) = 4 |
| min | Listenin minumum ögesini döndürür. | min(L1) = 1 |

Sıralı Tipler - Tuple

Tuple, listeye birçok yönden benzer. Farklı veri türlerinin öğelerinin koleksiyonunu içerir. Demetin öğeleri virgülle (,) ayrılır ve parantez () içine alınır. Öğelerinin boyutunu ve değerini değiştiremediğimiz için salt okunur bir veri yapısıdır.

```
tup = ("Merhaba", "Python", 2)
```

```
print (type(tup))    # tipini yazdırma  
print (tup)          #tuple yazdırma
```

```
print (tup[1:])       #dilimleme  
print (tup[0:1])
```

```
print (tup + tup)     #ucuca ekleme  
print (tup * 3)       #tekrarlama
```

```
tup[2] = "Safak"      # deger ekleme.. !! HATA
```

Tuple indeksleme ve bölme

Tuple = (0, 1, 2, 3, 4, 5)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|

Tuple[0] = 0 Tuple[0:] = (0, 1, 2, 3, 4, 5)

Tuple[1] = 1 Tuple[:] = (0, 1, 2, 3, 4, 5)

Tuple[2] = 2 Tuple[2:4] = (2, 3)

Tuple[3] = 3 Tuple[1:3] = (1, 2)

Tuple[4] = 4 Tuple[:4] = (0, 1, 2, 3)

Tuple[5] = 5

Sırasız Tipler - Set

Set içerisindeki her öğe benzersiz (tekil), değişmez olmalıdır ve kümeler yinelenen öğeleri kaldırır. Setin elemanlarına eklenmiş bir index yoktur, yani, dizinin herhangi bir öğesine doğrudan erişilemez.

Kümelerin kendisi değiştirilebilir, yani oluşturulduktan sonra onu değiştirebiliriz.

set() fonksiyonu kullanılarak oluşturulur veya bir dizi öğe küme ayraçları içinde koyularak virgülle ayrılır. Çeşitli türde değerler içerebilir.

Not : Bos set {} ile yaratılmaz

```
set1 = set()
```

```
set2 = {'Safak', 2, 3, 'Ahmet'}
```

```
print(set2)
```

```
set2.add(10)      # Eleman ekleme
```

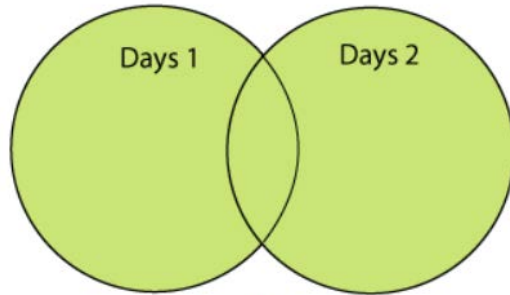
```
print(set2)
```

```
set2.remove(2)    # Eleman silme
```

```
print(set2)
```

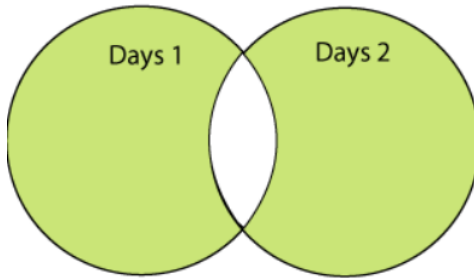
Set İşlemleri

Birleşim



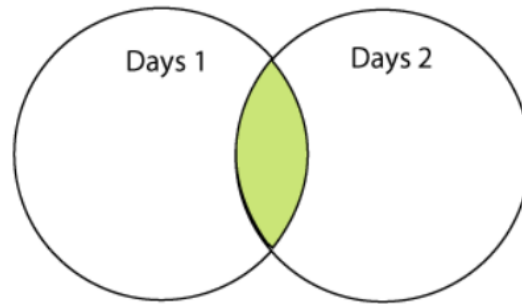
union | operator
union() method

Simetrik Fark



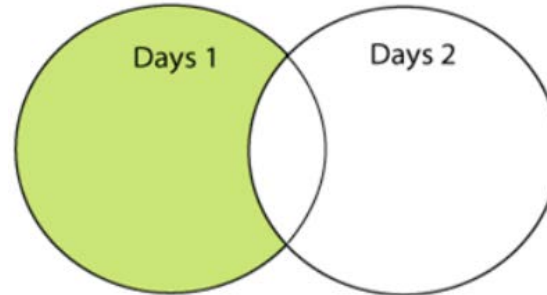
^ operator
symmetric_difference() method

Kesişim



& operator
intersection() method
intersection_update()

Fark



(-) operator
difference() method

```
Days1 = {"Pazartesi", "Salı", "Çarşamba",  
        "Perşembe"}  
Days2 = {"Pazartesi", "Salı", "Cumartesi"}
```

```
print(Days1 | Days2)  
print(Days1.union(Days2))
```

```
print(Days1 & Days2)  
print(Days1.intersection(Days2))
```

```
print(Days1 - Days2)  
print(Days1.difference(Days2))
```

```
print(Days1 ^ Days2)  
print(Days1.symmetric_difference(Days2))
```

Set fonksiyonları

| | | |
|---|--------------------------|---|
| 1 | <u>add(item)</u> | Sete bir öğe ekler. Öğe sette zaten mevcutsa hiçbir etkisi yoktur. |
| 2 | clear() | Setteki tüm öğeleri siler. |
| 3 | copy() | Setin kopyasını döndürür. |
| 4 | difference_update(...) | Belirtilen setlerde bulunan tüm öğeleri kaldırarak bu seti değiştirir. |
| 5 | <u>discard(item)</u> | Setten belirtilen öğeyi kaldırır. |
| 6 | intersection() | Her iki kümenin yalnızca ortak öğelerini içeren yeni bir küme döndürür. |
| 7 | intersection_update(...) | Her iki sette de bulunmayan öğeleri orijinal setten kaldırır. |
| 8 | Isdisjoint(...) | İki kümede boş bir kesişme varsa True döndür. |

Set fonksiyonları

| | | |
|----|---|---|
| 9 | <code>Issubset(...)</code> | Başka bir setin bu seti içerip içermediğini bildirir. |
| 10 | <code>Issuperset(...)</code> | Bu setin başka bir set içerip içermediğini bildirir. |
| 11 | <code><u>pop()</u></code> | Remove and return an arbitrary set element that is the last element of the set. Raises <code>KeyError</code> if the set is empty. |
| 12 | <code><u>remove(item)</u></code> | Kümenin son öğesi olan rastgele bir set öğesini kaldırın ve geri döndürür. Küme boşsa <code>KeyError</code> 'ı verir. |
| 13 | <code>symmetric_difference(...)</code> | verilen kümelerin simetrik farkını alır |
| 14 | <code>symmetric_difference_update(...)</code> | verilen kümelerin simetrik farkını alır ve günceller |
| 15 | <code>union(...)</code> | Kümelerin birleşimi yapar |
| 16 | <code>update()</code> | Bir seti kendisinin ve diğerlerinin birleşimiyle günceller. |

Sırasız Tipler - Dictionary

Sözlük, bir anahtar-değer çiftinden oluşan sırasız bir kümedir. Her bir anahtarın belirli bir değeri depoladığı ilişkilendirilebilir bir tablo gibidir.

Anahtar herhangi bir ilkel veri türünü tutabilir; değer ise herhangi bir Python nesnesi olabilir.

Sözlükteki öğeler virgülle (,) ve her anahtar- değerinden iki nokta üst üste (:) ile ayrılır ve süslü parantez {} içinde olurlar.

```
d = {1:'Ahmet', 2:'Mehmet', 3:'Serap', 4:'Meral'}
```

```
print (d)
```

```
print("İlk isim: "+d[1])
```

```
print("4. isim: "+ d[4])
```

```
print (d.keys())
```

```
print (d.values())
```

dict () metodu ile de oluşturulabilir.

```
dict = dict({"birinci": 'Java', "ikinci": 'Python',  
"ucuncu": 'C++'})
```

```
print(dict)
```

```
print(dict["ikinci"])
```

```
for x in dict:
```

```
    print(x,dict[x])
```

Dictionary fonksiyonları

| | | |
|----|---|--|
| 1 | <u>dic.clear()</u> | Sözlüğün tüm öğelerini silmek için kullanılır |
| 2 | <u>dict.copy()</u> | Sözlüğün sıfır bir kopyasını döndürür. |
| 3 | <u>dict.fromkeys(iterable, value = None, /)</u> | Değere eşit değerlerle yinelenenlerden yeni bir sözlük oluşturun. |
| 4 | <u>dict.get(key, default = "None")</u> | verilen anahtar için belirtilen değeri almak için kullanılır. |
| 5 | <u>dict.has_key(key)</u> | belirtilen anahtarı varsa true döndürür. |
| 6 | <u>dict.items()</u> | Tüm anahtar / değer çiftlerini bir tuple olarak döndürür. |
| 7 | <u>dict.keys()</u> | Sözlüğün tüm anahtarlarını döndürür. |
| 8 | <u>dict.setdefault(key, default= "None")</u> | Anahtar sözlükte belirtilmemişse anahtarı varsayılan değere ayarlamak için kullanılır. |
| 9 | <u>dict.update(dict2)</u> | Bu sözlüğe dict2 anahtar / değer çiftini ekleyerek sözlüğü günceller. |
| 10 | <u>dict.values()</u> | Sözlüğün tüm değerlerini döndürür. |
| 11 | <u>pop()</u> | verilen anahtar değerine göre değeri kaldırır. |
| 12 | <u>popitem()</u> | rastgele bir öğeyi kaldırır. |

Sırasız Tipler - Boolean

Boolean türü, True ve False olmak üzere iki yerleşik değer sağlar. Bu değerler, verilen ifadenin doğru veya yanlış olduğunu belirlemek için kullanılır. Bool sınıfıyla ifade eder.

```
print(type(True))  
print(type(False))  
print(false)
```

Python Programlama

Koşullu İfadeler

DR. ŞAFAK KAYIKÇI

Koşullu İfadeler

Karar verme, belirli bir koşul için belirli bir kod bloğunu çalıştırmamıza izin verir.

Python'da karar verme aşağıdaki ifadelerle gerçekleştirilir.

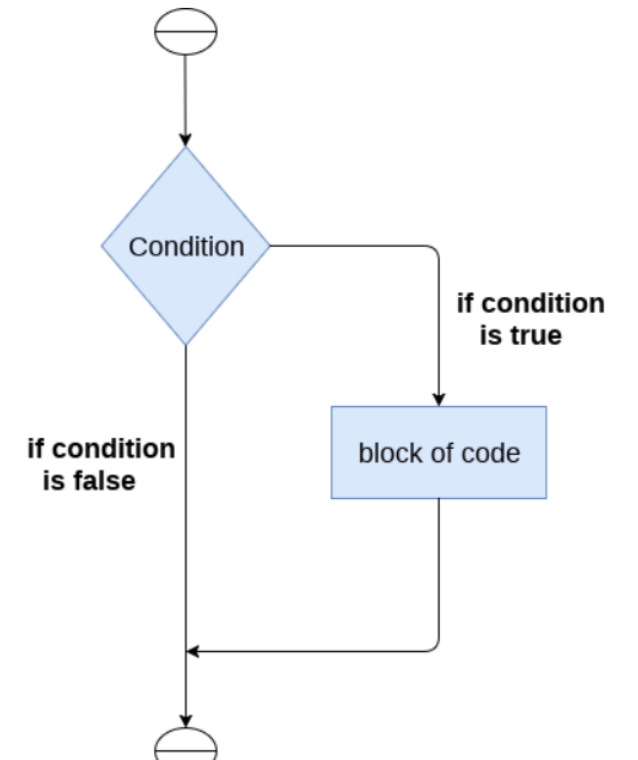
| İfade | Açıklama |
|-----------|---|
| If | If ifadesi, belirli bir koşulu test etmek için kullanılır. Koşul doğruysa, bir kod bloğu (if-blok) yürütülür. |
| If - else | If-else ifadesi, kontrol edilecek koşulun yanlış durumu için kod bloğu sağlaması dışında if ifadesine benzer. If ifadesinde sağlanan koşul yanlışsa, else ifadesi çalıştırılacaktır |
| İç içe if | İç içe geçmiş ifadeler, if ? Else ifadelerinin bir dış if ifadesinin içinde kullanılmasını sağlar. |

if ifadesi

İf ifadesi, belirli bir koşulu test etmek için kullanılır ve koşul doğruysa, if-bloğundaki bir kod yürütür. İf ifadesinin koşulu, doğru veya yanlış olarak değerlendirilebilen herhangi bir geçerli mantıksal ifade olabilir.

```
if expression:  
    statement
```

```
num = int (input("Bir sayi giriniz:"))  
if num%2==0:  
    print("Sayi cifttir")
```

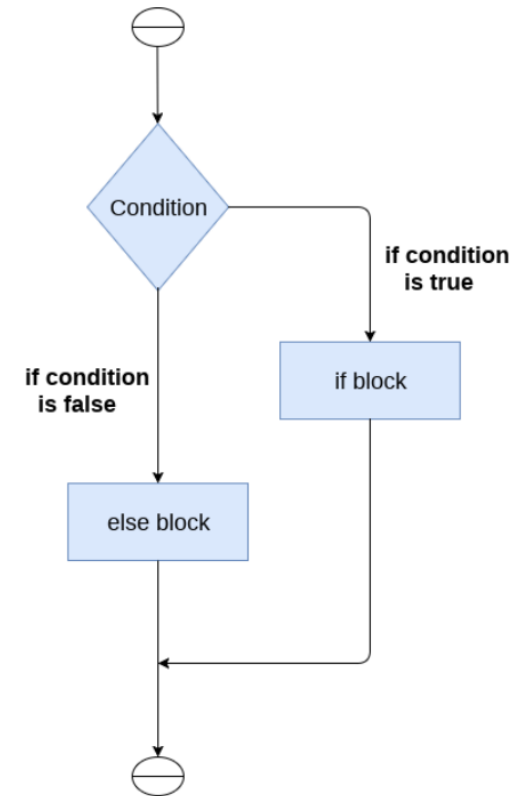


if-else ifadesi

İf-else ifadesi, koşulun yanlış olması durumunda çalıştırılan if ifadesiyle birleştirilmiş bir else bloğu sağlar. Koşul doğruysa, if bloğu yürütülür, değilse else-blok yürütülür.

```
if condition:  
    #block of statements  
else:  
    #another block of statements (else-block)
```

```
num = int (input("Bir sayi giriniz:"))  
if num%2==0 :  
    print("Sayi cifttir")  
else:  
    print("sayi tektir")
```

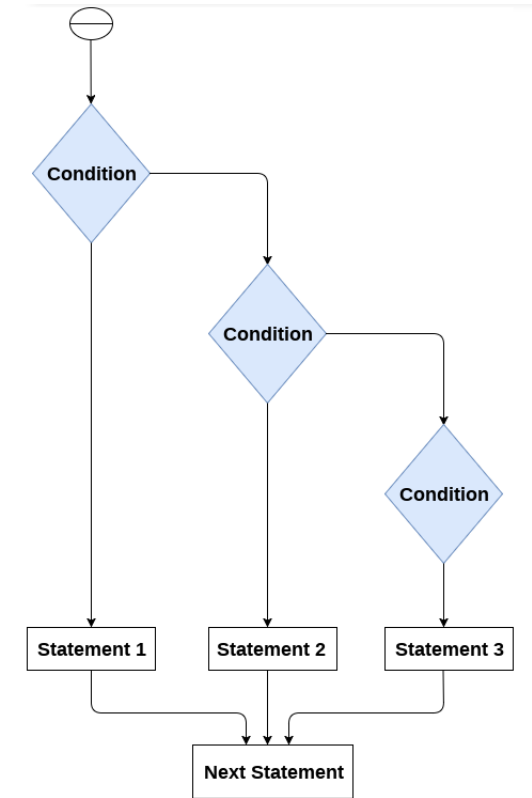


elif ifadesi

elif ifadesi, birden çok koşulun kontrol edilmesi ve aralarındaki gerçek koşula bağlı olarak belirli bir ifade bloğunun yürütülmesini sağlar. C'deki else if komutu ile aynıdır.

```
sinavNotu = int(input("Final notunu giriniz:"))  
if sinavNotu>85 and sinavNotu<=100:  
    print("Final notunuz:A")  
elif sinavNotu>65 and sinavNotu<=85:  
    print("Final notunuz:C")  
elif sinavNotu>50 and sinavNotu<=65:  
    print("Final notunuz:D")  
else:  
    print("Kaldiniz...")
```

```
if expression 1:  
    # block of statements  
elif expression 2:  
    # block of statements  
elif expression 3:  
    # block of statements  
else:  
    # block of statements
```



Pyhton Programlama Döngüler

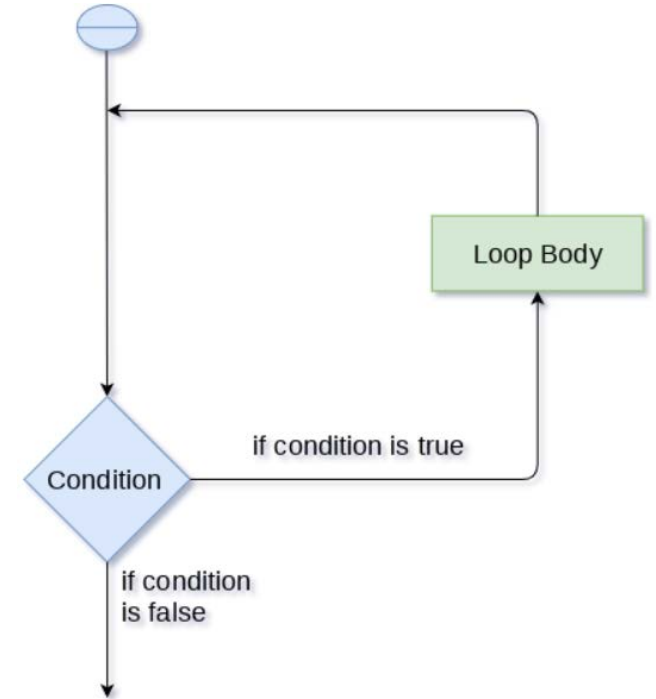
DR. ŞAFAK KAYIKÇI

Döngüler

Belirli bir kodu birkaç kez tekrarlamak için döngüler kullanılır.

Döngü tipleri :

1. for döngüsü
2. while döngüsü
3. do..while döngüsü

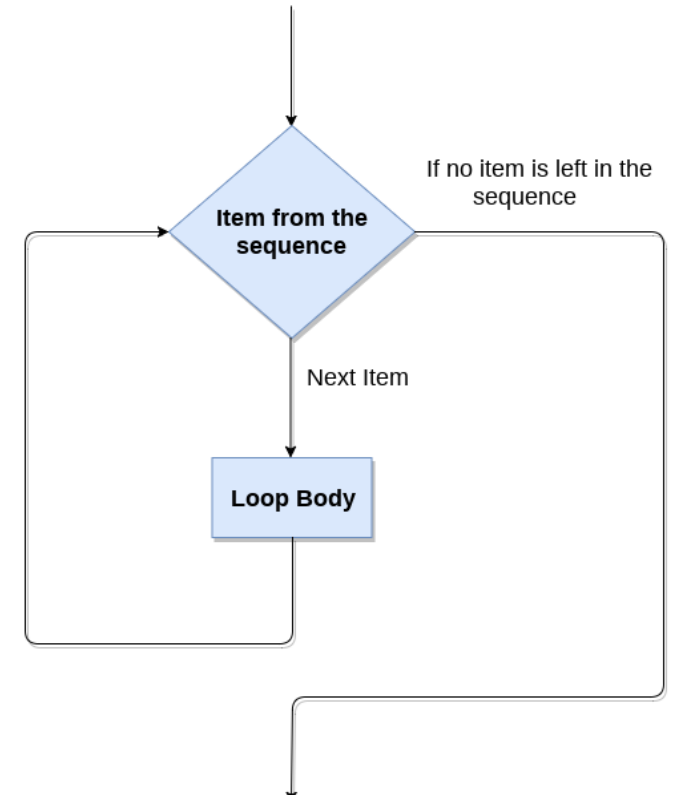


for döngüsü

for döngüsü , ifadeleri veya programın bir bölümünü birkaç kez yinelemek için kullanılır. Sıklıkla liste, tuple veya sözlük gibi veri yapılarında gezinmek için kullanılır.

```
str = "Safak"  
for i in str:  
    print(i)
```

```
list=[1,2,3,4,5]  
toplam=0  
for i in list:  
    toplam+=i  
    print(toplam)
```



range kavramı

range() fonksiyonu sayı dizisi üretmek için kullanılır.

```
range(start,stop,step size)
```

1. start, iterasyonun başlangıcını temsil eder.
2. stop, döngünün stop kadar yineleneyeceğini gösterir. range (1,5) yazımı 1 ila 4 yineleme iterasyon yaratacaktır. İsteğe bağlıdır.
3. step size, iterasyondaki belirli sayıları atlamak için kullanılır. Varsayılan olarak adım boyutu 1'dir. İsteğe bağlıdır.

```
for i in range(10):  
    print(i)
```

```
for i in range(2,10,2):  
    print(i)
```

```
list = ['Ali','Veli','Ayse','Mine']  
for i in range(len(list)):  
    print("Selam",list[i])
```

İç içe döngüler

for döngüsü içine herhangi bir sayıda for döngüsü yerleştirilebilir . İç döngü, dış döngünün her yinelenmesi için n sayıda yürütülür.

```
for iterating_var1 in sequence: #outer loop
    for iterating_var2 in sequence: #inner loop
        #block of statements
    #Other statements
```

```
for i in range(5):
    for j in range(5):
        print(i,j)
```

For döngüsü ile else ifadesi

C, C++ veya Java gibi diğer dillerin aksine Python, else ifadesini yalnızca tüm yinelemeler tükendiğinde çalıştırılabilen for döngüsü ile kullanabilir. Eğer döngü break ifadelerinden herhangi birini içeriyorsa, else ifadesinin çalıştırılmaz.

```
for i in range(0,5):  
    print(i)  
else:  
    print("for döngüsü tamamen çalıştı..break yok")
```

break ifadesi nedeniyle döngü kopmuştur;
bu nedenle, else ifadesi çalıştırılmayacaktır.



```
for i in range(0,5):  
    print(i)  
    break;  
else:print("for dongusu calisti");  
print("Break oldugundan dolayi donguden cikti...")
```

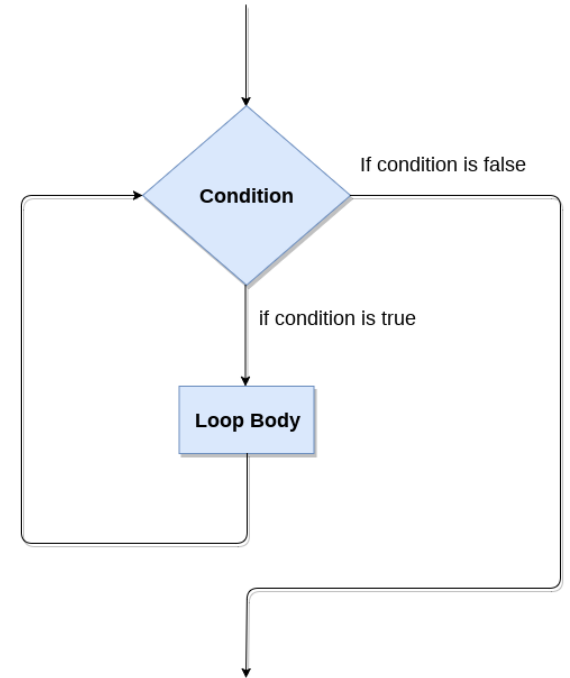
while döngüsü

Verilen koşul yanlış döndürene kadar kod çalıştırılır. Aynı zamanda önceden test edilmiş bir döngü olarak da bilinir.

```
while expression:  
    statements
```

```
str='SAFAK'  
i=0  
while i < len(str):  
    if str[i] == 'A':  
        i+=1  
        continue  
    print(str[i])  
    i+=1
```

```
str='SAFAK'  
i=0  
while i < len(str):  
    if str[i] == 'F':  
        i+=1  
        break  
    print(str[i])  
    i+=1
```



while döngüsü ile else ifadesi

Python, else ifadesini while döngüsü ile birlikte kullanılabilir. While ifadesinde verilen koşul yanlış olduğunda else bloğu çalıştırılır. For döngüsü gibi, while döngüsü break deyimi kullanılarak bozulursa, o zaman else bloğu çalıştırılmayacak ve else bloğundan sonra bulunan ifade çalıştırılacaktır.

```
i=1
while(i<=5):
    print(i)
    i=i+1
else:
    print("while dongusu calisti")
```

```
i=1
while(i<=5):
    print(i)
    i=i+1
    if (i==3):
        break
else:
    print("while dongusu calisti")
```

Python Programlama

String

DR. ŞAFAK KAYIKÇI

String

Python'da string, tek tırnak işaretleri, çift tırnak işaretleri veya üçlü tırnak işaretleri ile çevrili karakterler dizisidir.

```
str1 = 'Safak'  
str2 = "Safak"  
str3 = """ Üçlü tırnaklar genellikle  
        çoklu satırlar için kullanılır """
```

String

indekslenmesi 0'dan başlar. Örneğin, "HELLO" dizisi aşağıdaki şekilde gösterildiği gibi indekslenir.

str = "HELLO"

| | | | | |
|----------|----------|----------|----------|----------|
| H | E | L | L | O |
| 0 | 1 | 2 | 3 | 4 |

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

Bölümleme

str = "HELLO"

| | | | | |
|---|---|---|---|---|
| H | E | L | L | O |
| 0 | 1 | 2 | 3 | 4 |

| | |
|--------------|-------------------|
| str[0] = 'H' | str[:] = 'HELLO' |
| str[1] = 'E' | str[0:] = 'HELLO' |
| str[2] = 'L' | str[:5] = 'HELLO' |
| str[3] = 'L' | str[:3] = 'HEL' |
| str[4] = 'O' | str[0:2] = 'HE' |
| | str[1:4] = 'ELL' |

str = "HELLO"

| | | | | |
|----|----|----|----|----|
| H | E | L | L | O |
| -5 | -4 | -3 | -2 | -1 |

| | |
|---------------|---------------------|
| str[-1] = 'O' | str[-3:-1] = 'LL' |
| str[-2] = 'L' | str[-4:-1] = 'ELL' |
| str[-3] = 'L' | str[-5:-3] = 'HE' |
| str[-4] = 'E' | str[-4:] = 'ELLO' |
| str[-5] = 'H' | str[::-1] = 'OLLEH' |

String değer değiştirme

```
str = "HELLO"  
str[0] = "h"  
print(str)
```

Traceback (most recent call last):

File "12.py", line 2, in <module>

str[0] = "h";

TypeError: 'str' object does not support item assignment

```
del str[1]
```

TypeError: 'str' object doesn't support item deletion

Stringler, immutable (değiştirilmez) yapılardır.

Ancak yeni bir içeriğe tamamen atanabilir.

```
str = "HELLO"  
print(str)  
str = "hello"  
print(str)
```

String operatörleri

| Operator | Açıklama |
|----------|---|
| + | Operatörün her iki tarafında verilen dizeleri birleştirmek için kullanılan birleştirme operatörü olarak bilinir. |
| * | Tekrar operatörü olarak bilinir. Aynı dizenin birden çok kopyasını birleştirir. |
| [] | Dilim operatörü olarak bilinir. Belirli bir dizenin alt dizelerine erişmek için kullanılır. |
| [:] | Aralık dilim operatörü olarak bilinir. Belirtilen aralıktaki karakterlere erişmek için kullanılır. |
| in | Üyelik operatörü olarak bilinir. Belirtilen dizede belirli bir alt dizge varsa döndürür. |
| not in | Aynı zamanda bir üyelik operatörüdür ve in'in tam tersini yapar. Belirtilen dizede belirli bir alt dize yoksa true döndürür. |
| % | Dize biçimlendirmesi yapmak için kullanılır. Değerlerini python'da eşlemek için% d veya% f gibi C programlamasında kullanılan biçim belirleyicilerinden yararlanır. |

format methodu

format () metodu dizeleri biçimlendirme en esnek ve kullanışlı bir yöntemdir. Küme parantezleri {} dizede yer tutucu olarak kullanılır ve format () yöntemi argümanı ile değiştirilir.

süslü parantez kullanımı

```
print("{} ve {} haftasonu günleridir".format("Cumartesi","Pazar"))
```

#Konumsal kullanım

```
print("{1} ve {0} iyi transferler. ".format("Pelkas","Özil"))
```

#Keyword kullanımı

```
print("Mühendislikler: {a},{b},{c}".format(a = "Bilgisayar", b = "Elektrik/Elektronik", c = "Makine"))
```


% kullanarak format

Python, C'nin printf deyiminde kullanılan biçim belirleyicilerini kullanılmasına izin verir. Python'daki biçim belirleyicileri, C'de ele alındıkları gibi ele alınır. Bununla birlikte, Python, biçim belirleyicileri ve değerleri arasında bir arabirim olarak kullanılan ek bir% operatörü sağlar.

```
x = 10
y = 1.567
str = "Safak"
print("x: %d \ny:%f \nstr:%s"%(x,y,str))
```

String fonksiyonları

`capitalize()`

`casefold()`

`center(width ,fillchar)`

`count(string,begin,end)`

`decode(encoding = 'UTF8', errors = 'strict')`

`encode()`

`endswith(suffix
,begin=0,end=len(string))`

`expandtabs(tabsize = 8)`

`find(substring ,beginIndex,
endIndex)`

`format(value)`

`index(substring, beginIndex,
endIndex)`

`isalnum()`

`isalpha()`

`isdecimal()`

`isdigit()`

`isidentifier()`

`islower()`

`isnumeric()`

`isprintable()`

`isupper()`

`isspace()`

`istitle()`

`isupper()`

`join(seq)`

`len(string)`

`ljust(width[,fillchar])`

`lower()`

`lstrip()`

`partition()`

`maketrans()`

`replace(old,new[,count])`

`rfind(str,beg=0,end=len(str))`

`rindex(str,beg=0,end=len(str))`

`rjust(width,[,fillchar])`

`rstrip()`

`rsplit(sep=None, maxsplit = -1)`

`split(str,num=string.count(str))`

`splitlines(num=string.count("\n"))`

`startswith(str,beg=0,end=len(str))`

`strip([chars])`

`swapcase()`

`title()`

`translate(table,deletechars = "")`

`upper()`

`zfill(width)`

`rpartition()`

Pyhton Programlama Fonksiyonlar

DR. ŞAFAK KAYIKÇI

Fonksiyonlar

Fonksiyonlar, gerektiğinde çağrılabilen, yeniden kullanılabilir kodun organize bloğu olarak tanımlanabilir.

1. User-defined functions (Kullanıcı tanımlı fonksiyonlar)
2. Built-in functions (Yerleşik, önceden tanımlı fonksiyonlar)

- Fonksiyon tanımlamak için **def** anahtar sözcüğünü kullanılır
- Fonksiyon ismi, adlandırma kurallarına uymalıdır
- İsteğe bağlı olarak parametre alabilir
- Fonksiyon bloğu iki nokta üst üste (:) ile başlatılır ve blok ifadeleri aynı girintide olmalıdır.
- **return** ifadesi dönüş değeri döndürmek için kullanılır. Bir işlevin yalnızca bir dönüşü olabilir

```
def my_function(parameters):  
    function_block  
    return expression
```

Fonksiyon Çağırılması

Fonksiyonun çağırılmadan önce tanımlanmış olması gerekmektedir.

```
def merhaba():  
    print("Merhaba Dünya")  
merhaba()
```

```
def toplam():  
    a=10;b=20;  
    c=a+b  
    return c  
print(toplam())
```

return'un bir ifadesi yoksa veya işlevde kendisi yoksa, o zaman **None** nesnesini döndürür .

Parametreler:

```
def merhaba(isim):  
    print("Merhaba",isim)  
merhaba("Safak")
```

```
def toplam(a,b):  
    return a+b  
print(toplam(3,5))
```

Call by Reference

Python'da, referansla çağırım, gerçek değeri fonksiyona bir argüman olarak iletmek anlamına gelir. Tüm fonksiyonlar referans olarak çağrılır, yani işlevin içindeki referansta yapılan tüm değişiklikler, referans tarafından atıfta bulunulan orijinal değere geri döner.

Immutable Object (List)

```
def change(liste):  
    liste.append(20)  
    liste.append(30)  
    print("Fonksiyon ici:",liste)
```

```
liste1=[10,30,40]  
change(liste1)  
print("Fonksiyon dışı:",liste1)
```

Mutable Object (List)

```
def change (str):  
    str = "Merhaba " + str  
    print("Fonksiyon ici:",str)
```

```
string1 = "SAFAK"  
change(string1)  
print("Fonksiyon dışı:",string1)
```

Parametre Türleri

1. Gerekli (required) argümanlar
2. Varsayılan (default) argümanlar
3. Değişken uzunluklu (variable-length) argümanlar
4. Anahtar kelime (keyword) argümanlar

Gerekli (required) - Varsayılan (default)

```
def toplam(a,b):  
    return a+b  
toplam(10)
```

`TypeError: toplam() missing 1 required positional argument: 'b'`

```
def selamla(isim,sehir="Bolu"):  
    print(isim, "merhaba.",sehir," da mısınız")
```

```
selamla(isim = "Safak")  
selamla(sehir="Ankara",isim = "Safak")
```


variable-length arguments (*args)

Önceden iletilecek argümanların sayısı bilinmeyen durumlarda, Python fonksiyon çağrısında tuple olarak virgülle ayrılmış değerleri sunma esnekliği sağlar. Bu sayede herhangi bir sayıda parametre gönderilebilir.

Fonksiyon tanımında, değişken uzunluk argümanını * args kullanarak * <değişken - isim> olarak tanımlanır.

```
def yazdir(*kadro):  
    print("Gönderilen parametre tipi ",type(kadro))  
    for isim in kadro:  
        print(isim)  
yazdir("Altay","Gustavo","Pelkas","Sosa")
```

keyword arguments(**kwargs)

Anahtar kelime argümanları ile parametrelerin rastgele sırayla geçirilmesi sağlanabilir. **** kwargs** olarak tanımlanarak birden çok anahtar kelime argümanını iletilebilir. * args ile benzerdir ancak argümanı sözlük biçiminde saklar.

```
def yazdir(**kadro):  
    print(kadro)  
yazdir(a="Altay")  
yazdir(ortaSaha="Ozil,Pelkas", forvet="Valencia")
```

Değişkenlerin kapsamı

Değişkenlerin kapsamları, değişkenin bildirildiği konuma bağlıdır. İki tür kapsam vardır:

Yerel (lokal) değişkenler

```
def yazdir():  
    mesaj = "Merhaba Dünya" #lokal değişken  
    print(mesaj)  
yazdir()  
print(mesaj) #error ! lokal değişkene buradan erişilemez
```

```
NameError: name 'mesaj' is not defined
```

Global değişkenler

```
def hesapla(*args):  
    toplam=0  
    for arg in args:  
        toplam = toplam +arg  
    print("toplam:",toplam)  
  
toplam=0  
hesapla(10,20,30) #sonuc 60 yazdırılacak  
print("fonksiyon dışında:",toplam) #sonuc 0 yazdırılacak
```

Lambda fonksiyonları

Lambda fonksiyonları, isimsiz olarak tanımlanan anonim fonksiyonlardır. Python, fonksiyonları standart yani `def` anahtar sözcüğünü kullanarak tanımlamak yerine, anonim olarak **lambda** anahtar sözcüğü kullanılarak bildirmeye izin verir.

Lambda fonksiyonları herhangi bir sayıda bağımsız değişkeni kabul edebilir, ancak ifade biçiminde yalnızca bir değer döndürebilir.

```
lambda arguments: expression
```

Örnek

```
x = lambda a:a+10  
#print(x) #<function <lambda> at 0x0000017F74A034C0>  
print("toplam = ",x(20))
```



```
def x(a):  
    return a+10  
print("toplam = ",x(20))
```

İki parametre

```
x = lambda a,b: a*b  
print("çarpım = ", x(20,10))
```



```
def x(a,b):  
    return a*b  
print("çarpım = ",x(20,10))
```

lambda filter() – lambda map()

filter () fonksiyonu , parametre olarak bir fonksiyon ve bir listeyi alır. Listenin tüm öğelerini filtrelemek için etkili bir yol sağlar. fonksiyonun True olarak değerlendirildiği yeni sırayı döndürür.

```
#tek sayı içeren tuple döndüren filter
liste = (1,2,3,4,5,6)
tekSayilar = tuple(filter(lambda x:(x%2 != 0),liste))
print(tekSayilar)
```

map () fonksiyonu , parametre olarak bir fonksiyon ve bir listeyi alır. Her öğe için işlev tarafından döndürülen tüm değiştirilmiş öğeleri içeren yeni bir liste verir.

```
# tuple, lambda işlevinin doğru olarak değerlendirdiği listenin tüm öğelerini içerir
liste = (10,20,30,40,50,60)
karesi = tuple(map(lambda x:x**2,liste))
print(karesi)
```

Pyhton Programlama Modüller

DR. ŞAFAK KAYIKÇI

Modüller

Modüller fonksiyon, sınıf veya değişkenleri içeren bir python program kod dosyası .py olarak tanımlanabilir. Kodu mantıklı bir şekilde düzenleme esnekliği sağlar. Bir modülün diğerinde kullanmak için, belirli modülün içe aktarılması gerekir.

1. import statement

```
import module1,module2,..... module n
```

2. from-import statement

```
from < module-name> import <name 1>, <name 2>..,<name n>
```


Örnek

merhaba.py

```
def selamla(name):  
    print("Merhaba "+name);
```

1. import statement

| | |
|------------------------|------------------------|
| import merhaba | import merhaba as mrb |
| isim=input("Adınız:") | isim=input("Adınız:") |
| merhaba.selamla(isim) | mrb.selamla(isim) |

2. from-import statement

```
from merhaba import selamla  
isim=input( "Adınız:")  
selamla(isim)
```

dir() fonksiyonu

dir () fonksiyonu, aktarılan modülde tanımlanan isimlerin sıralı bir listesini döndürür. Bu liste, bu modülde tanımlanan tüm alt modülleri, değişkenleri ve fonksiyonları içerir.

```
import json
List = dir(json)
print(List)
```

```
['JSONDecodeError', 'JSONDecoder', 'JSONEncoder', '__all__', '__author__', '__builtins__', '__cached__',  
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '__version__',  
'_default_decoder', '_default_encoder', 'codecs', 'decoder', 'detect_encoding', 'dump', 'dumps', 'encoder',  
'load', 'loads', 'scanner']
```

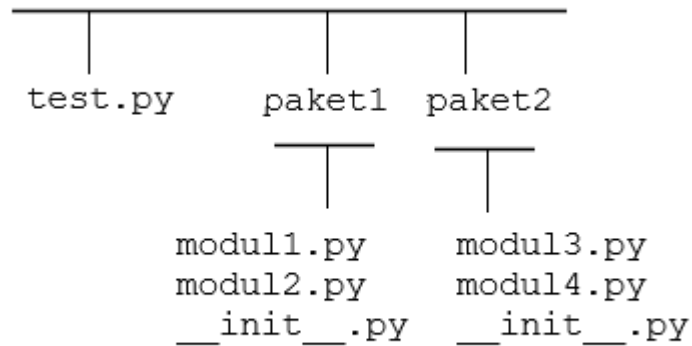
reload() fonksiyonu

Bir modül, python kaynak dosyasına içe aktarılma sayısına bakılmaksızın **bir kez** yüklenir. Bununla birlikte, üst düzey kodu yeniden çalıştırmak için önceden içe aktarılmış modülü yeniden yüklemek istenirse bize reload () fonksiyonu sağlanabilir.

```
reload(<module-name>)
```

Paketler

Paketler, içindeki alt paketler, modüller ve alt modüller ile hiyerarşik bir dizin yapısı sağlayarak geliştiriciyi uygulama geliştirme ortamı ile kolaylaştırır. Uygulama kodunu verimli bir şekilde sınıflandırmak için kullanılır.



modül1

```
def print1_1():
    print("Paket1... Modül1")
.....diğer modüller.....
```

__init__.py

```
from .modul1 import print1_1
from .modul2 import print1_2
```

test.py

```
import paket1,paket2
paket1.print1_1()
paket2.print2_4()
```

Python Programlama

File I/O

DR. ŞAFAK KAYIKÇI

File I/O

Python'da dosyalar metin veya ikili (binary) olarak iki modda ele alınır. Dosya metin veya ikili formatta olabilir ve bir dosyanın her satırı özel karakterle biter.

İşlem sırası :

1. Dosya açma
2. Okuma veya yazma
3. Dosya kapatma

Dosya Açmak

```
file object = open(<file-name>, <access-mode>, <buffering>)
```

bir dosya nesnesi döndürür

read (okuma), write(yazma), append (ekleme)

```
# dosya.txt dosyasını okuma modunda açar  
fileptr = open("deneme.txt", "r")  
  
if fileptr:  
    print("Dosya açıldı")
```

| | | |
|----|-----|--|
| 1 | r | Dosyayı salt okunur modunda açar. File pointer başlangıçta bulunur. Erişim modu yazılmazsa dosya varsayılan olarak bu modda açılır. |
| 2 | rb | Dosyayı binary biçimde salt okunur olarak açar. File pointer dosyanın başlangıcında bulunur. |
| 3 | r+ | Dosyayı hem okumak hem de yazmak için açar. File pointer dosyanın başlangıcında bulunur. |
| 4 | rb+ | Dosyayı hem ikili biçimde okumak hem de yazmak için açar. File pointer dosyanın başlangıcında bulunur. |
| 5 | w | Dosyayı sadece yazılacak şekilde açar. Önceden varsa dosyanın üzerine yazar veya aynı ada sahip bir dosya yoksa yeni bir tane oluşturur. File pointer dosyanın başlangıcında bulunur. |
| 6 | wb | Dosyayı yalnızca ikili biçimde yazacak şekilde açar. Önceden varsa dosyanın üzerine yazar veya dosya yoksa yeni bir tane oluşturur. File pointer dosyanın başlangıcında bulunur. |
| 7 | w+ | yazmak ve okumak için dosyayı açar. r+ önceden yazılmış dosyanın üzerine yazmaz, bu açıdan r+ 'dan farklıdır. Dosya yoksa yeni bir dosya oluşturur. File pointer dosyanın başlangıcında bulunur. |
| 8 | wb+ | ikili biçimde yazmak ve okumak için dosyayı açar. File pointer dosyanın başlangıcında bulunur. |
| 9 | a | Dosyayı ekleme modunda açar. File pointer , varsa önceden yazılmış dosyanın sonunda bulunur. Aynı ada sahip bir dosya yoksa yeni bir dosya oluşturur. |
| 10 | ab | Dosyayı ikili biçimde ekleme modunda açar. File pointer , önceden yazılmış dosyanın sonunda bulunur. Aynı ada sahip bir dosya yoksa, ikili biçimde yeni bir dosya oluşturur. |
| 11 | a+ | eklemek ve okumak için bir dosya açar. Bir dosya varsa, File pointer dosyanın sonunda kalır. Aynı ada sahip bir dosya yoksa yeni bir dosya oluşturur. |
| 12 | ab+ | ikili biçimde eklemek ve okumak için bir dosya açar. File pointer dosyanın sonunda kalır. |

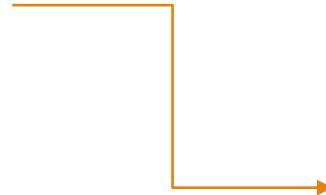
Dosya Kapatmak

Dosya üzerinde tüm işlemler tamamlandıktan sonra, `close ()` yöntemini kullanarak kapatılmalıdır. Bir dosya nesnesinde `close ()` yöntemi çağrıldığında yazılmayan bilgiler kaybolacaktır.

```
# dosya.txt dosyasını okuma modunda açar  
fileptr = open("deneme.txt","r")
```

```
if fileptr:  
    print("Dosya açıldı")
```

```
#açılmış dosyayı kapatır  
fileptr.close()
```



```
try:  
    fileptr = open("deneme.txt","r")  
  
    if fileptr:  
        print("Dosya açıldı")  
  
finally:  
    fileptr.close()
```

with komutu

with deyimini kullanmanın avantajı, iç içe geçmiş bloğun nasıl çıktığına bakılmaksızın dosyayı kapatma garantisi sağlamasıdır. kod bloğunda break, return veya istisna oluşursa dosyayı otomatik olarak kapatır. close () fonksiyonunu yazmak gerekmez ve dosyanın bozulmasını engeller.

```
with open(<file name>, <access mode>) as <file-pointer>:  
    #statement suite
```

```
with open("deneme.txt",'r') as f:  
    content = f.read();  
    print(content)
```

Dosyaya Yazmak

Bir dosyaya biraz metin yazmak için, dosyayı aşağıdaki erişim modlarından biriyle açma yöntemini kullanarak açılması gerekmektedir:

w: Herhangi bir dosya varsa dosyanın üzerine yazacaktır. File pointer dosyanın başlangıcındadır. Dosya yoksa yeni bir dosya oluşturur.

a: Mevcut dosyanın sonuna ekleyecektir. File pointer dosyanın sonundadır. Dosya yoksa yeni bir dosya oluşturur.

```
fileptr = open("deneme", "a")  
  
fileptr.write("\nMerhaba Safak")  
  
fileptr.close()
```

Dosyayı Okumak

Python read () komutu ile metin verileri veya binary dosyaları okunur.

```
fileobj.read (<count>)
```

dosyanın başından başlayarak dosyadan okunacak bayt sayısıdır.
sayı belirtilmezse dosyanın içeriğini sonuna kadar okuyabilir.

```
fileptr = open("deneme.txt", "r")  
content = fileptr.read(10)  
print(type(content))  
print(content)  
fileptr.close()
```

Satır okumak – readline(), readlines()

read yok

```
#read yok  
fileptr = open("deneme.txt","r")  
for i in fileptr:  
    print(i) # i dosyanın her satırını içerir  
fileptr.close()
```

readline()

```
fileptr = open("deneme.txt","r");  
  
content = fileptr.readline()  
content1 = fileptr.readline()  
content2 = fileptr.readline()  
print(content)  
print(content1)  
print(content2)  
  
fileptr.close()
```

readlines()

```
fileptr = open("deneme.txt","r");  
#dosyanın tüm verilerini değişken içeriğinde depolar  
content = fileptr.readlines()  
print(content)  
fileptr.close()
```

File Pointer (Dosya İşaretçisi)

tell() fonksiyonu, dosya işaretçisinin o anda mevcut olduğu bayt numarasını gösterir.

```
fileptr = open("deneme.txt", "r")
#başlangıç durumu 0
print("Başlangıç pozisyonu :", fileptr.tell())
content = fileptr.read();
print("Okuma sonrası pozisyon:", fileptr.tell())
```

dosya işaretçisi konumunu değiştirmek için seek () fonksiyonu kullanılır.

dosya işaretçisinin
dosya içindeki yeni
konumu

```
<file-ptr>.seek(offset[, from])

fileptr = open("deneme.txt", "rb")
#başlangıç durumu 0
print("Başlangıç pozisyonu :", fileptr.tell())
fileptr.seek(10);
print("Şu anki pozisyon:", fileptr.tell())
fileptr.seek(-3, 1);
print("Şu anki pozisyon:", fileptr.tell())
fileptr.seek(-5, 2);
print("Şu anki pozisyon:", fileptr.tell())
```

Baytların taşınacağı referans konumunu gösterir.

0: dosyanın başlangıcı

1: dosya işaretçisinin geçerli konumu

2: dosya işaretçisinin sonu

Python Programlama Exception - Assert

DR. ŞAFAK KAYIKÇI

Exception (İstisnalar)

İstisnalar, programın akışında kesinti ile sonuçlanan olağandışı bir durumlardır. Bir istisna meydana geldiğinde, program yürütmeyi durdurur ve bu nedenle diğer kod çalıştırılmaz.



Örnek

Çoklu İstisnalar


```
try:
    a = int(input("Birinci sayı:"))
    b = int(input("İkinci sayı:"))
    c = a/b

    print("a/b:%d" %c)

except :
    print("Sıfıra bölünemez")
except Exception as e:
    print("Sıfıra bölünemez")
    print(e)

else:
    print("İşlem başarılı")

finally:
    print("Her zaman çalışır..")
```



```
try:
    fileptr = open("file.txt","r")
    a = int(input("Birinci sayı:"))
    b = int(input("İkinci sayı:"))
    c = a/b

    print("a/b:%d" %c)
```

1. **except** ArithmeticError:
print("Sıfıra bölünemez")
 2. **except** IOError:
print("IOError")
- ```
else:
 print("İşlem başarılı")

finally:
 print("Her zaman çalışır..")
```

# İstisna Fırlatma

---

Python'da **raise** anahtar kelimesi kullanılarak bir istisna fırlatılabilir. Programın yürütülmesini durdurmak için bir istisna oluşturmamız gereken durumlarda kullanışlıdır.

```
raise Exception_class,<value>
```

```
try:
 age = int(input("Yaşınızı giriniz:"))
 if(age<18):
 raise ValueError
 else:
 print("Oy kullanabilirsiniz")
except ValueError:
 print("Oy kullanamazsınız")
```

```
try:
 age = int(input("Yaşınızı giriniz:"))
 if(age<18):
 raise ValueError("Oy kullanamazsınız")
 else:
 print("Oy kullanabilirsiniz")
except ValueError as e:
 print(e)
```

# Özel İstisnalar

---

Python'da kendi istisnalarımızda sınıf olarak yaratabiliriz.

```
class MyException(Exception):
 def __init__(self, data):
 self.data = data
 def __str__(self):
 return repr(self.data)

try:
 raise MyException(2000)
except MyException as e:
 print("Hata Durumu:", e.data)
```

# Assert

---

Python **assert** anahtar sözcüğü, bir koşulu test eden bir debug aracı olarak tanımlanabilir. Programda bir olguyu güvenle iddia eden veya ifade eden varsayımlardır.

Koşulun doğru mu yoksa yanlış mı döndürdüğünü kontrol eden bir Boolean değeridir. Doğruysa, program hiçbir şey yapmaz ve bir sonraki kod satırına geçer. Yanlışsa, isteğe bağlı bir hata mesajı ile bir **AssertionError** istisnası oluşturur.

Uygulama geliştirme alanındaki test veya kalite güvencesi için önemlidir.

```
assert condition, error_message(optional)
```

# Örnek

```
def ortalama(vizeler):
 assert len(vizeler) != 0, "Liste boş"
 return sum(vizeler)/len(vizeler)

notlar1 = [67,59,86,75,92]
print("Notlar1 ortalaması:",ortalama(notlar1))

notlar2 = []
print("Notlar2 ortalaması:",ortalama(notlar2))
```



Notlar1 ortalaması: 75.8  
Traceback (most recent call last):

**AssertionError:** Liste boş

```
x = 5
y = 0
print ("x / y : ")
assert y != 0, "Sıfıra Bölünemez"
print (x / y)
```



x / y :  
Traceback (most recent call last):

**AssertionError:** Sıfıra Bölünemez

# Pyhton Programlama Loglama

---

DR. ŞAFAK KAYIKÇI

# Logging

---

Log dosyaları, işletim sistemi, yazılım veya networkte meydana gelen çeşitli olaylarla ilgili bilgileri içerir. Log kaydı, sadece yazılım geliştirmedeki hataları tanımlamakla sınırlı değildir. Ayrıca güvenlik olaylarını tespit etmede, politika ihlallerini izlemeye, sorun olması durumunda bilgi sağlamada, uygulama darboğazlarını bulmada veya kullanım verilerini oluşturmada kullanılır.

Loglamanın amaçları:

- Bilgi toplama
- Sorun giderme
- İstatistikler oluşturma
- Arşivleme
- Profil oluşturma
- Debug

# Neler Loglanır - Loglanmaz

---

Loglanması gereken olaylar, giriş doğrulama hatalarını, kimlik doğrulama ve yetkilendirme hatalarını (authentication and authorization), uygulama hatalarını, konfigürasyon değişikliklerini ve uygulama başlatma ve kapatmaları bilgilerini içerir.

Loglanmaması gereken olaylar arasında uygulama kaynak kodu, oturum tanımlama değerleri, erişim belirteçleri (Access tokens), hassas kişisel veriler, parolalar, veritabanı bağlantı stringleri, şifreleme anahtarları, banka hesabı ve kart sahibi verileri bulunur.



# Özellikleri

---

- Loglama anlamlı olmalıdır.
- Loglama içeriği belli olmalıdır.
- Loglama farklı seviyelerde yapılandırılmalı ve yapılmalıdır.
- Loglama dengeli olmalıdır; çok az veya çok fazla bilgi içermemelidir.
- Günlüğe kaydetme mesajları insanlar tarafından anlaşılabilir ve makineler tarafından ayrıştırılabilir olmalıdır.
- Daha karmaşık uygulamalarda oturum açma, birkaç günlük dosyasında yapılmalıdır.
- Loglama, geliştirme (development) ve canlı ortama (production) uyarlanmalıdır.

# Logging modülü

---

Python logging modülü, uygulamalar ve kütüphaneler için esnek bir olay günlük kaydı sistemi uygulayan fonksiyonları ve sınıfları tanımlar.

logging modülünün dört ana bileşeni vardır:

1. Loggers : uygulama kodunun doğrudan kullandığı arabirimi ortaya çıkarır.
2. Handlers : günlük kayıtlarını (loggers tarafından oluşturulan) uygun hedefe gönderir.
3. Filters: hangi günlük kayıtlarının çıkarılacağını belirlemek için daha ince bir kolaylık sağlar.
4. Formatters : son çıktıdaki günlük kayıtlarının düzenini belirtir.

# Log seviyeleri

---

1. **NOTSET**
2. **DEBUG** - Ayrıntılı bilgi sağlamak ve yalnızca sorunları teşhis ederken kullanmak için kullanılır.
3. **INFO** - İşlerin istediğimiz gibi çalıştığına dair bilgi sağlar.
4. **WARNING** - Beklenmedik bir şey olduğunda uyarmak için kullanılır, dikkate alınmazsa problemle ilerleyen zamanda karşılaşılabılır.
5. **ERROR** - Ciddi bir sorun olduğunda, yazılımın bazı programları çalıştırmadığını bildirmek için kullanılır.
6. **CRITICAL** - Ciddi hatayı belirtir, programın kendisi çalıştırılamayabilir.

| Level    | Sayısal Değer |
|----------|---------------|
| NOTSET   | 0             |
| DEBUG    | 10            |
| INFO     | 20            |
| WARNING  | 30            |
| ERROR    | 40            |
| CRITICAL | 50            |

# Root Logger

---

Tüm loggerlar, root loggerın türemiştir. Her logger, log mesajlarını üstüne iletir. Yeni loggerlar `getLogger(name)` fonksiyonu ile oluşturulur. Fonksiyonun isimsiz `getLogger()` çağırılması, root logger döndürür.

Root loggerın varsayılan log seviyesi WARNING dir.

Root logger, hiyerarşinin en üstünde yer alır ve yapılandırılmamış olsa bile her zaman mevcuttur. Genel olarak, program veya kütüphane doğrudan kök kaydediciye karşı oturum açmamalıdır. Bunun yerine program için belirli bir logger yapılandırılmalıdır. Root logger, tüm kitaplıklardan tüm loggerları kolayca açıp kapatmak için kullanılabilir.

# basit bir örnek

---

```
import logging

logging.debug('debug mesajı')
logging.info('info mesajı')
logging.warning('warning mesajı')
logging.error('error mesajı')
logging.critical('critical mesajı')

WARNING:root:warning mesajı
ERROR:root:error mesajı
CRITICAL:root:critical mesajı

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

print(logger.getEffectiveLevel())
```

varsayılan olarak log mesajları WARNING, ERROR ve CRITICAL önem seviyesiyle kaydedilir.

# Dosyaya yazdırmak

---

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)

handler = logging.FileHandler('deneme.log')
handler.setLevel(logging.INFO)

logging format oluşturun
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
handler.setFormatter(formatter)

handler'i logger a kaydedin
logger.addHandler(handler)

logger.info('Merhaba Python')
```

# Config dosyası

---

```
import logging
import logging.config

logging.config.fileConfig(fname='log.conf')

logger = logging.getLogger('dev')
logger.info('Bu bir bilgilendirme mesajıdır')
```

## log.conf

```
[loggers]
keys=root,dev

[handlers]
keys=consoleHandler

[formatters]
keys=extend,simple

[logger_root]
level=INFO
handlers=consoleHandler

[logger_dev]
level=INFO
handlers=consoleHandler
qualname=dev
propagate=0

[handler_consoleHandler]
class=StreamHandler
level=INFO
formatter=extend
args=(sys.stdout,)

[formatter_extend]
format=%(asctime)s - %(name)s - %(levelname)s - %(message)s

[formatter_simple]
format=%(asctime)s - %(message)s
```

# Pyhton Programlama

## Date - Time

---

DR. ŞAFAK KAYIKÇI



# datetime

---

Python'da date bir veri türü değildir, ancak **datetime**, **time** ve **calendar** modülleri ile tarih nesneleriyle çalışılabilir.

datetime sınıfları altı ana sınıflarına ayrılır :

**date** – ham tarih. Yıl, ay, gün özelliklerine sahiptir.

**time** - Her günün tam olarak  $24 * 60 * 60$  saniyeye sahip olduğunu varsayarak ince ayar zaman verir. Nitelikler olarak saat, dakika, saniye, mikrosaniye ve tzinfo'ya sahiptir .

**datetime** - Yıl, ay, gün, saat, dakika, saniye, mikrosaniye ve tzinfo özellikleriyle birlikte tarih ve saat gruptandır.

**timedelta** - İki tarih, saat veya tarih saat arasındaki farkı ile mikrosaniye çözünürlük seviyesinde indirger.

**tzinfo** - Saat dilimi bilgi nesneleri sağlar.

**timezone** - Python'un yeni sürümüne dahildir. Bu, tzinfo soyut temel sınıfını uygulayan sınıftır..

# Tick

---

Python'da, zaman anları 1 Ocak 1970 12: 00'den itibaren sayılır . time() fonksiyonu , 1 Ocak 1970, 00: 00'dan bu yana harcanan toplam tik sayısını döndürür. Bir tik, zamanı ölçen en küçük birim olarak görülebilir.

```
import time;
12 AM, 1st January 1970 beri geçen tick
print(time.time())
1618412205.1382227
```

# Time Tuple

---

```
#time tuple döndürür
print(time.localtime(time.time()))
```

```
time.struct_time(tm_year=2021, tm_mon=4, tm_mday=14, tm_hour=17, tm_min=59, tm_sec=23,
tm_wday=2, tm_yday=104, tm_isdst=0)
```

| Index | Attribute        | Values                     |
|-------|------------------|----------------------------|
| 0     | Year             | 4 digit (for example 2018) |
| 1     | Month            | 1 to 12                    |
| 2     | Day              | 1 to 31                    |
| 3     | Hour             | 0 to 23                    |
| 4     | Minute           | 0 to 59                    |
| 5     | Second           | 0 to 60                    |
| 6     | Day of weak      | 0 to 6                     |
| 7     | Day of year      | 1 to 366                   |
| 8     | Daylight savings | -1, 0, 1 , or -1           |

Formatlanmış zaman için asctime kullanılır: `print(time.asctime(time.localtime(time.time())))`

# Datetime Modülü

---

Datetime modülü, özel bir tarih nesneleri oluşturulabilir ve karşılaştırma vb. gibi tarihlerde çeşitli işlemleri gerçekleştirilebilir.

Tarihlerle tarih nesneleri olarak çalışmak için , datetime modülünü python kaynak koduna aktarmamız gerekir.

```
import datetime

print(datetime.datetime.now())
print(datetime.datetime(2020,4,4))
```

>, >=, <Ve <= gibi karşılaştırma operatörlerini kullanarak iki tarihi karşılaştırabilir.

```
if 8<datetime.datetime.now().hour<16:
 print("Mesai saatleri..")
else:
 print("Tatil saatleri..")
```

# Takvim modülü

---

Python, takvimlerle çalışmak için çeşitli yöntemler içeren bir **calendar** nesnesi sağlar.

```
import calendar
cal = calendar.month(2021,4)
print(cal)
```

`calendar.prcal(2021)` tüm yılı bastıracaktır.

```
April 2021
Mo Tu We Th Fr Sa Su
 1 2 3 4
 5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

# Pyhton Programlama

## Nesne Tabanlı Programlama

---

DR. ŞAFAK KAYIKÇI

# Nesne Tabanlı Programlama

---

Nesneye yönelik bir paradigma, programı sınıfları ve nesneleri kullanarak tasarlamaktır. Nesneler oluşturarak sorunu çözmek yaygın bir tekniktir. Yeniden kullanılabilir kod yazmaya odaklanır.

**Inheritance (Kalıtım):** Kalıtım, gerçek dünyadaki kalıtım kavramını simüle eden nesne yönelimli programlamanın en önemli yönüdür. Alt nesnenin, ana nesnenin tüm özelliklerini ve davranışlarını edindiğini belirtir. Kalıtımı kullanarak, başka bir sınıfın tüm özelliklerini ve davranışını kullanan bir sınıf oluşturulabilir. Yeni sınıf, türetilmiş sınıf veya alt sınıf olarak bilinir ve özellikleri edinilen sınıf, temel sınıf veya ana sınıf olarak bilinir. Kodun yeniden kullanılabilirliğini sağlar.

**Polimorfizm (Çok biçimlilik):** Polimorfizm ile, bir görevin farklı şekillerde gerçekleştirilebilme imkanı sağlanır.

**Encapsulation (Kapsülleme):** Yöntemlere ve değişkenlere erişimi kısıtlamak için kullanılır. Kapsüllemeye, kod ve veriler tek bir birim içinde birbirine sarmalanır.

**Soyutlama (Abstraction):** Soyutlama ve kapsülleme genellikle eş anlamlılar olarak kullanılır. Her ikisi de neredeyse eş anlamlıdır çünkü veri soyutlaması kapsülleme yoluyla elde edilir. Soyutlama, dahili ayrıntıları gizlemek ve yalnızca işlevleri göstermek için kullanılır.

# Class (Sınıf) – Object (Nesne)

---

Sınıf, nesnelerin bir koleksiyonu olarak tanımlanabilir. Bazı belirli niteliklere ve yöntemlere sahip mantıksal bir varlıktır. Sınıf sanal bir varlıktır ve bir nesnenin bir taslağı olarak görülebilir. Sınıf, somutlaştırıldığında nesne oluşur.

Nesne, durumu ve davranışı olan bir varlıktır. Python'daki her şey bir nesnedir ve neredeyse her şeyin nitelikleri ve yöntemleri vardır. Tüm işlevlerin yerleşik bir `__doc__` özniteliği vardır ve kaynak kodunda tanımlanan docstring'i döndürür.

**del** anahtar sözcüğünü kullanarak nesnenin kendisi veya nesnenin özellikleri silinebilir.



# self parametresi

---

**self** , geçerli sınıf nesnesine atıfta bulunan bir referans değişkeni olarak kullanılır. Sınıfın geçerli örneğini ifade eder ve sınıf değişkenlerine erişir. Self yerine her şey kullanılabilir ancak sınıfa ait olan herhangi bir fonksiyonun ilk parametresi olmalıdır. Ancak fonksiyon çağırımlarında kullanımı opsiyoneldir.

```
class araba:
 def __init__(self,model, yil):
 self.model = model
 self.yil = yil
 def display(self):
 print(self.model,self.yil)
```

```
a1 = araba("Opel", 2020)
a1.display()
```

```
del a1.yil
a1.display()
```

```
del a1
a1.display()
```

# Constructor (yapılandırıcı)

---

Yapılandırıcılar, sınıflardan nesne oluşturmak için kullanılan özel bir fonksiyon türüdür. Yapılandırıcılar ayrıca nesnenin herhangi bir başlatma görevini gerçekleştirmesi için yeterli kaynak olduğunu doğrular. C ++ veya Java'da, yapılandırıcılar sınıfla aynı isme sahiptir, ancak Python'da farklı şekilde kullanılır.

Parametreler iki tipte olabilir:

1. Parametrelili Yapılandırıcı
2. Parametresiz Yapılandırıcı

# Constructor (yapılandırıcı)

---

`__init__()` fonksiyonu , sınıf yapıcısını simüle eder. Bu yöntem, sınıf başlatıldığında çağrılır. `self` anahtar sözcüğünü, sınıfın özniteliklerine veya yöntemine erişime izin veren bir ilk argüman olarak kabul eder .

`__init__()` tanımına bağlı olarak, sınıf nesnesini yaratma anında herhangi bir sayıda argüman iletilebilir . Çoğunlukla sınıf özniteliklerini ilk değer vermek için kullanılır. Varsayılan yapılandırıcıya bağlı olsa bile her sınıfın bir yapılandırıcısı olmalıdır.

```
class araba:
 def __init__(self,model, yıl):
 self.model = model
 self.yıl = yıl
 def display(self):
 print(self.model,self.yıl)
```

```
a1 = araba("Opel", 2020)
a2= araba ("Toyota",2018)

a1.display()
a2.display()
```

# yerleşik (built-in) sınıf fonksiyonları

---

| SN | Fonksiyon                 | Açıklama                                                                     |
|----|---------------------------|------------------------------------------------------------------------------|
| 1  | getattr(obj,name,default) | Nesnenin özniteliğine erişmek için kullanılır.                               |
| 2  | setattr(obj, name,value)  | Bir nesnenin belirli özelliğine belirli bir değer ayarlamak için kullanılır. |
| 3  | delattr(obj, name)        | Belirli bir niteliği silmek için kullanılır.                                 |
| 4  | hasattr(obj, name)        | Nesne belirli bir nitelik içeriyorsa true döndürür                           |

# yerleşik (built-in) sınıf fonksiyonları

---

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

#Öğrenci sınıfının nesnesini oluşturur
a = Araba("Opel", 2002)

nesnenin öznitelik adını yazdırır
print(getattr(a, 'model'))

#özellik yil degerini 2015 yapar
setattr(a, 'model', 2015)

#değiştirilen model değerini yazdırır
print(getattr(a, 'model'))

prints true if the student contains the attribute with name id
print(hasattr(a, 'id'))

yıl özelliğini siler
delattr(a, 'yil')

yıl özelliği silindiğinden bu bir hata verecektir
print(a.yil)
```

# yerleşik (built-in) sınıf öznitelikleri

---

| SN | Öznitelik               | Açıklama                                                  |
|----|-------------------------|-----------------------------------------------------------|
| 1  | <code>__dict__</code>   | Sınıf ad alanıyla ilgili bilgileri içeren sözlüğü sağlar. |
| 2  | <code>__doc__</code>    | Sınıf belgelerine sahip bir dize içerir                   |
| 3  | <code>__name__</code>   | Sınıf adına erişmek için kullanılır.                      |
| 4  | <code>__module__</code> | Bu sınıfın tanımlandığı modüle erişmek için kullanılır.   |
| 5  | <code>__bases__</code>  | Tüm temel sınıfları içeren bir demet içerir.              |

# yerleşik (built-in) sınıf öznitelikleri

---

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

 def display_details(self):
 print("Model:%s, yil:%d"%(self.model,self.yil))

a = Araba("Ford",2018)
print(a.__doc__)
print(a.__dict__)
print(a.__module__)
```

# Inheritance (kalıtım)

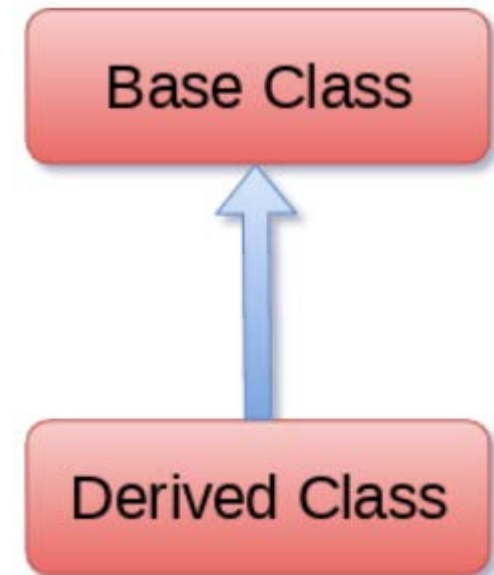
Kalıtımda, alt sınıf (child class) ana sınıftan (parent class) türetilir ve ana sınıfın özellikleri ve metotlarına sahip olur. Programda kodun yeniden kullanılabilirliğini sağlar çünkü bir sınıfı sıfırdan oluşturmak yerine mevcut bir sınıf kullanılabilir.

```
class derived-class(base class):
 <class-suite>
```

```
class Hayvan:
 def sesCikar(self):
 print("Hayvan ses cikardi...")
```

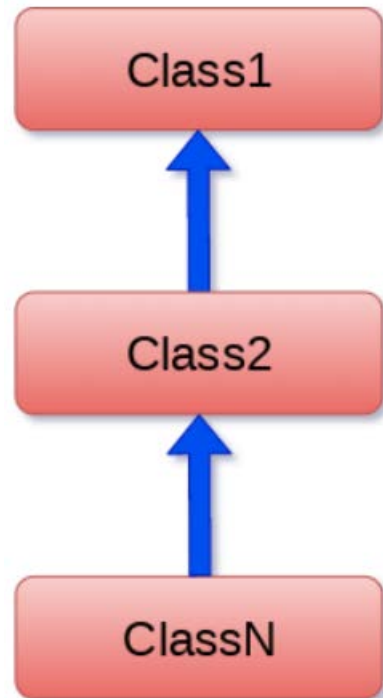
```
class Kopek(Hayvan):
 def havla(self):
 print("hav hav...")
```

```
k = Kopek()
k.havla()
k.sesCikar()
```





# Multi-Level Inheritance



```
class class1:
 <class-suite>
class class2(class1):
 <class suite>
class class3(class2):
 <class suite>
.
```

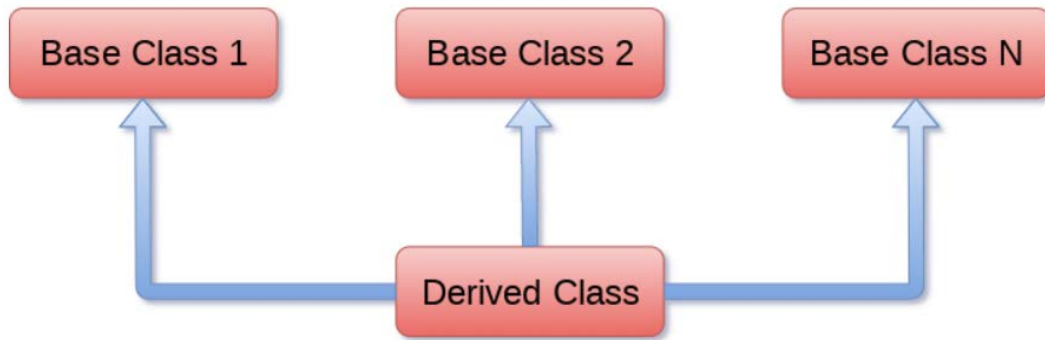
```
class Hayvan:
 def sesCikar(self):
 print("Hayvan ses cikardi...")

class Kopek(Hayvan):
 def havla(self):
 print("hav hav...")

class KurtKopegi(Kopek):
 def yemekYe(self):
 print("Kurtkopegi yemek yedi")

k = KurtKopegi()
k.sesCikar()
k.havla()
k.yemekYe()
```

# Multiple inheritance



```
class Base1:
 <class-suite>

class Base2:
 <class-suite>
.
.
.
class BaseN:
 <class-suite>

class Derived(Base1, Base2, BaseN):
 <class-suite>
```

```
class Hayvan:
 def sesCikar(self):
 print("Hayvan ses cikardi...")
```

```
class Beslenme:
 def turBelirt(self):
 print("Etobur..")
```

```
class Kopek(Hayvan):
 def havla(self):
 print("hav hav...")
```

```
class KurtKopeci(Kopek, Beslenme):
 def yemekYe(self):
 print("Kurtkopeci yemek yedi")
```

```
k = KurtKopeci()
k.sesCikar()
k.havla()
k.yemekYe()
k.turBelirt()
```

# issubclass(sub,sup) - isinstance (obj, class)

---

issubclass (sub, sup) yöntemi, belirtilen sınıflar arasındaki ilişkileri kontrol etmek için kullanılır. Birinci sınıf ikinci sınıfın alt sınıfı ise true, aksi takdirde false döndürür.

isinstance () yöntemi, nesneler ve sınıflar arasındaki ilişkiyi kontrol etmek için kullanılır. Birinci parametre, yani obj, ikinci parametrenin, yani sınıfın örneğiye, true döndürür.

```
class Hayvan:
 def sesCikar(self):
 print("Hayvan ses cikardi...")

class Beslenme:
 def turBelirt(self):
 print("Etobur..")

class Kopek(Hayvan):
 def havla(self):
 print("hav hav...")

class KurtKopegi(Kopek,Beslenme):
 def yemekYe(self):
 print("Kurtkopegi yemek yedi")

k = KurtKopegi()

print(issubclass(KurtKopegi,Hayvan))
print(isinstance(k,Hayvan))
```

# veri soyutlama (abstraction)

---

Python'da, gizlenecek özneliğe örnek olarak çift alt çizgi (\_\_) ekleyerek de veri gizleme yapabilir. Bundan sonra, nitelik nesne aracılığıyla sınıfın dışında görünmeyecektir.

```
class Araba:
 __count = 0;
 def __init__(self):
 Araba.__count = Araba.__count+1
 def yazdir(self):
 print("Uretilen araba sayisi",Araba.__count)

a1 = Araba()
a2 = Araba()
try:
 print(a1.__count)
finally:
 a1.yazdir()
```

# abstract class

---

- Bir veya daha fazla soyut yöntemden oluşan sınıfa soyut sınıf (abstract class) denir.
- Soyut sınıflar hem normal yöntemler hem de soyut yöntemler içerebilir. Soyut yöntemler uygulamalarını içermez.
- Soyut sınıf, alt sınıf tarafından miras alınabilir ve soyut yöntem tanımını alt sınıfta yapılır.
- Soyut sınıf için nesneler yaratılmaz.
- Soyut bir yöntemi tanımlamak için @abstractmethod dekoratörü kullanılır veya yönteme tanımı sağlanmazsa, bu otomatik olarak soyut yöntem olur.
- Diğer dillerin aksine, Python soyut sınıfın kendisini sağlamaz. Python da soyutlamayı kullanmak için (abstract base class başharfleri) abc modülü kullanılır.

```
from abc import ABC

class ClassName(ABC):
```

# abstract class

---

```
from abc import ABC, abstractmethod

class Hayvan(ABC):

 @abstractmethod
 def sesCikar(self):
 pass

class Kedi(Hayvan):
 def sesCikar(self):
 print("Miyav..")

class Kopek(Hayvan):
 def sesCikar(self):
 print("Havhav..")

class Kus(Hayvan):
 def sesCikar(self):
 print("Cikcik....")

kedi = Kedi()
kedi.sesCikar()

kopek = Kopek()
kopek.sesCikar()

kus = Kus()
kus.sesCikar()
```

# Pyhton Programlama mySQL Bağlantısı

---

DR. ŞAFAK KAYIKÇI

# mysql.connector

---

Python uygulamasını MySQL veritabanına bağlamak için **mysql.connector** modülünü kullanılmalıdır. mysql.connector, python kurulumuyla birlikte gelen yerleşik bir modül değildir.

```
pip install mysql-connector-python
```



# Veritabanı Bağlantısı

---

Python uygulamasını veritabanına bağlanması için aşağıdaki adımlar uygulanır:

1. `mysql.connector` modülünü içe aktarılır (import)

2. **connection** nesnesi oluşturulur

```
Connection-Object = mysql.connector.connect (host = <host-name>, user = <username>, passwd = <password>)
```

3. **cursor** (imleç) nesnesi oluşturulur

```
<my_cur> = conn.cursor()
```

4. sorgu çalıştırılır

# Veritabanı Bağlantısı

---

```
import mysql.connector

#connction nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "test",
 auth_plugin='mysql_native_password')

print(myconn)

#cursor nesnesi oluşturma
cur = myconn.cursor()
print(cur)
```

# Veritabanı Oluşturma

---

```
create database <database-name>
```

```
import mysql.connector

#connction nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "test",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()

try:
 cur.execute("create database PythonDB")
 dbs = cur.execute("show databases")
except:
 myconn.rollback()
for x in cur:
 print(x)

myconn.close()
```

# Tablo oluşturma

---

create table calisanlar (isim varchar(20) not null, id int primary key, maas float not null, dept\_Id int not null)

```
import mysql.connector

#connection nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "pythondb",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()

try:
 dbs = cur.execute('''create table calisanlar (isim varchar(20) not null,
 id int primary key,
 maas float not null, dept_Id int not null)''')

except:
 myconn.rollback()

myconn.close()
```

# Alter table

---

```
import mysql.connector

#connction nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "pythondb",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()

try:
 cur.execute("alter table calisanlar add lokasyon varchar(20) not null")
except:
 myconn.rollback()

myconn.close()
```

# Insert Table

---

```
import mysql.connector

#connction nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "pythondb",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()
sql = '''insert into calisanlar(isim, id, maas, dept_id, lokasyon)
 values (%s, %s, %s, %s, %s)'''

val = [("Safak", 102, 25000.00, 201, "Istanbul"),
 ("Murat", 103, 25000.00, 202, "Izmir"),
 ("Ayse", 104, 90000.00, 201, "Bolu")]

try:
 #tabloya kayıt ekleme
 cur.executemany(sql, val)
 #commit
 myconn.commit()
 print(cur.rowcount, "kayıt eklendi..")
except:
 myconn.rollback()

myconn.close()
```

# read (Select)

---

```
import mysql.connector

#connction nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "pythondb",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()

try:

 #calisanlar tablosundan select ile okuma
 cur.execute("select isim, id, maas from calisanlar where id in (102,103)")

 #butun kayıtları getirir. (fetchone tek kayıt getirir)
 result = cur.fetchall() → result = cur.fetchone()
 print(result)

 print("isim\t id \t maas\t");

 for row in result:
 print("%s\t %d\t %d\t"%(row[0],row[1],row[2]))
except:
 myconn.rollback()

myconn.close()
```

# Update - Delete

---

```
import mysql.connector

#connection nesnesi oluşturma
myconn = mysql.connector.connect(host = "localhost", user = "root",
 passwd = "root", database = "pythondb",
 auth_plugin='mysql_native_password')

#cursor nesnesi oluşturma
cur = myconn.cursor()

try:
 cur.execute("update calisanlar set isim = 'veli' where id = 102")
 cur.execute("delete from calisanlar where id = 103")
 myconn.commit()
except:
 myconn.rollback()

myconn.close()
```



# Pyhton Programlama Multithreading

---

DR. ŞAFAK KAYIKÇI

# Thread (iş parçacığı)

---

İş parçacığı (thread), bağımsız olarak veya işletim sistemi tarafından yürütülen, bir programın veya işlemin(process) en küçük birimidir.

İşletim Sistemi, işlemi (process) iş parçacıklarına (threads) bölerek çoklu görev (multitasking) gerçekleştirir. İş parçacığı, işlemlerin (process) sistem üzerinde ayrı ayrı yürütülmesini sağlayan 'lightweight process' olarak tanımlanabilir

Python 3'te, bir program üzerinde birden fazla işlemci çalışırken, her işlemci görevlerini ayrı ayrı yürütmek için aynı anda çalışır.

# Multithreading

---

Multithreading, CPU yardımı (bağlam değiştirme = context swithing) ile iş parçacıkları arasında hızla geçiş yaparak eşzamanlı olarak birden çok iş parçacığı çalıştırılmasıdır. Birden çok görevi aynı anda gerçekleştirmeyi sağlayarak performansı, hızı ve uygulamanın oluşturulmasını iyileştirmeyi hedefler.

Not: Python Global Yorumlayıcı Kilidi (Python Global Interpreter Lock – GIL), makinede birden fazla işlemciye sahip olsa bile, aynı anda tek bir iş parçacığı çalıştırılmasına izin verir.

# Pyhton - Multithreading

---

Python'da multithreading için **threading** modülü kullanılır.

```
import time
import threading

def thread1():
 time.sleep(3)
 print('Thread 1 çalışıyor')

def thread2(x):
 print('Thread 2 parametre ile çalışıyor: %d' %x)

if __name__ == '__main__':
 t1 = threading.Thread(target=thread1)
 t2 = threading.Thread(target=thread2,args=(10,))

 # start the threads
 t1.start()
 t2.start()

 # join the main thread
 t1.join()
 t2.join()
```

İş parçacıkları başladığında, mevcut program (main) da çalışmaya devam eder. Ana programın, iş parçacığı yürütmesi tamamlanana kadar çalışmasını tamamlamasını önlemek için join() yöntemi kullanıyoruz.Join () yöntemini kullanmazsak, yorumlayıcı Python programı içindeki herhangi bir print ifadesini çalıştırabilir. Genel olarak, yorumlayıcı kod satırlarını programın başlangıcından itibaren yürüttüğü için ilk print ifadesini yürütür.

# Thread modüle fonksiyonları

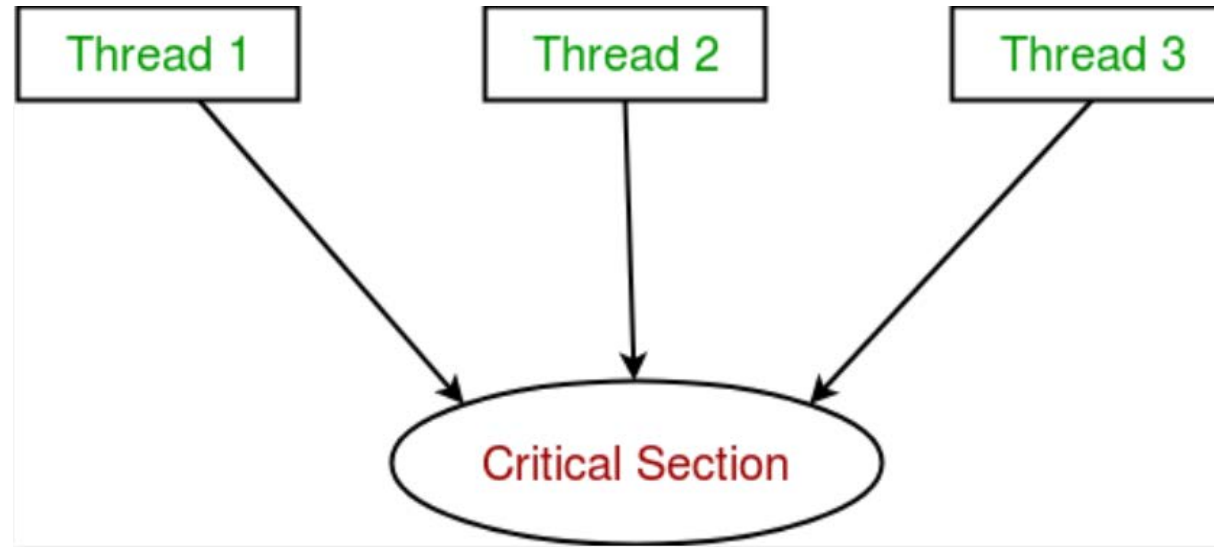
---

- **threading.active\_count()** : Çalışan Thread nesnesi sayısını döndürür
- **threading.current\_thread()** : O an çalışan Thread nesnesini döndürür
- **threading.get\_ident()** : Thread ID 'sini döndürür. Pozitif tamsayıdır.
- **threading.enumerate()** : O an canlı olan tüm Thread nesnelerin bir listesini döndürür . Liste daemon threadleri (program sona erdiğinde, onunla ilişkili tüm deamon threadler otomatik olarak öldürülür), mevcut thread tarafından oluşturulan dummy threadleri ve main threadi içerir. Henüz başlatılmamış ve sonlandırılmış threadler bu listede bulunmamaktadır.
- **threading.main\_thread()** : Ana Thread nesnesini döndürür
- **threading.stack\_size([size])** : Yeni Thread oluştururken kullanılan iş parçacığı yığın boyutunu döndürür. size değişkeni isteğe bağlıdır ve sonraki iş parçacıkları oluşturmak için kullanılacak yığın boyutunu ayarlamak için kullanılabilir. 0 veya pozitif bir tam sayı olmalıdır (D = varsayılan değer 0'dır). Eğer thread stack size değiştirmek desteklenmiyorsa RuntimeError hatası oluşur . Belirtilen yığın boyutu geçersizse ValueError hatası oluşur.
- **threading.TIMEOUT\_MAX**

# Thread Senkronizasyonu

---

İki Threadin, paylaşılan kaynaklara erişmek için program içinde belirli bir bölümü aynı anda yürütememesi sağlanmasıdır. Bu durum "critical section" olarak adlandırılır. Bu durumdan kaçınmak için bir yarış koşulu (race condition) kullanılır.



# Pyhton Programlama Tkinter

---

DR. ŞAFAK KAYIKÇI

# Tkinter

---

```
pip install tk
```

Python, masaüstü tabanlı uygulamalar için grafik kullanıcı arabirimi (GUI) oluşturmak için standart Tkinter kitaplığını kullanılır. Aşağıdaki adımlar izlenir :

1. tkinter modülü import edilir
2. ana uygulama penceresini oluşturulur
3. pencereye label, düğmeler, frame vb. gibi bileşenler eklenir
4. mainloop() fonksiyonu çağırılır



# Bileşenler (Widget - Component)

---

Button

Canvas

Checkbutton

Entry

Frame

Label

ListBox

Menubutton

Menu

Message

Radiobutton

Scale

Scrollbar

Text

Toplevel

Spinbox

PanedWindow

LabelFrame

MessageBox

# Tkinter Geometry

---

Tkinter geometrisi, bileşenlerin ekranda nasıl gözükeceğini belirtir.

1. `pack ()` yöntemi
2. `grid ()` yöntemi
3. `place ()` yöntemi

# pack() method

---

pack() yöntemi kullanılarak python uygulamasına eklenen konum bileşenleri, yöntem çağrısında belirtilen çeşitli seçenekler kullanılarak kontrol edilebilir. Bununla birlikte, kontroller daha azdır ve bileşenler genellikle daha az organize bir şekilde eklenir.

```
widget.pack(options)
```

- **expand:** expand true değerine ayarlanırsa, bileşen herhangi bir alanı dolduracak şekilde genişler.
- **fill:** Varsayılan olarak NONE olarak ayarlanmıştır. Ancak, bileşenin fazladan boşluk içerip içermediğini belirlemek için bu X veya Y olarak ayarlanabilir.
- **side:** pencere üzerinde bileşenin yerleştirileceği ebeveyn tarafını (yönünü) temsil eder.

# grid() method

---

grid() yöntemi bileşenleri tablo biçiminde düzenler. Yöntem çağrısındaki seçenekler olarak satırlar ve sütunlar belirtilebilir. Ayrıca bir bileşenin sütun genişliğini (genişlik) veya satır aralığında (yükseklik) belirtilebilir. Bileşenleri yerleştirmenin daha organize bir yoludur.

```
widget.grid(options)
```

- **column** : Bileşenin yerleştirileceği sütun numarasıdır. En soldaki sütun 0 ile temsil edilir.
- **columnspan** : Bileşenin genişliğidir. Sütunun genişletildiği sütun sayısını temsil eder.
- **ipadx, ipady** : Bileşen kenarlığının içine yerleştirilecek piksel sayısını temsil eder.
- **padx, pady** : Bileşen kenarlığının dışına yerleştirilecek piksel sayısını temsil eder.
- **row** : Bileşenlerin yerleştirileceği satır numarasıdır. En üstteki satır 0 ile temsil edilir.
- **rowspan** : Bileşenin yüksekliğidir, yani bileşenin genişletildiği satırın sayısıdır.
- **sticky** : Hücre bir bileşenden daha büyükse, bileşenin hücre içindeki konumunu belirtmek için yapışkan kullanılır. Bileşenin konumunu temsil eden yapışkan harflerin birleştirilmiş hali olabilir. N, E, W, S, NE, NW, NS, EW, ES olabilir.

# place() method

---

place() yöntemi, bileşenleri x ve y koordinatlarına göre düzenler.

```
widget.place(options)
```

**anchor:** Bileşenlerin konteyner içindeki tam konumunu temsil eder. Varsayılan değer (yön) NW'dir (sol üst köşe)

**bordermode:** Kenarlık türünün varsayılan değeri INSIDE dır. Sınırın içindeki üst öğenin yok sayılmasını varsayar. Diğer seçenek OUTSIDE dır.

**height, width:** piksel cinsinden yüksekliği ve genişliği ifade eder.

**relheight, relwidth:** Ana yüksekliğinin ve genişliğinin oranını gösteren 0.0 ile 1.0 arasında reel sayıdır.

**relx, rely:** Yatay ve dikey yönde ofset olan 0.0 ile 1.0 arasında reel sayıdır.

**x, y:** Piksellerdeki yatay ve dikey ofseti ifade eder.

# Özelliklerin Bazıları

## Button(parent, options)

|                  |            |
|------------------|------------|
| activebackground | Image      |
| activeforeground | justify    |
| Bd               | Padx       |
| Bg               | pady       |
| Command          | Relief     |
| Fg               | State      |
| Font             | Underline  |
| Height           | Width      |
| Highlightcolor   | Wraplength |

## checkboxbutton(master, options)

|                   |                |             |               |
|-------------------|----------------|-------------|---------------|
| activebackground  | height         | selectimage | <b>Method</b> |
| activeforeground  | highlightcolor | state       |               |
| bg                | image          | underline   |               |
| bitmap            | justify        | variable    |               |
| bd                | offvalue       | width       | deselect()    |
| command           | onvalue        | wraplength  | flash()       |
| cursor            | padx           |             | invoke()      |
| disableforeground | pady           |             | select()      |
| font              | relief         |             | toggle()      |
| fg                | selectcolor    |             |               |

## Entry (parent, options)

|                     |                   |
|---------------------|-------------------|
| bg                  | insertofftime     |
| bd                  | insertontime      |
| cursor              | insertwidth       |
| exportselection     | justify           |
| fg                  | relief            |
| font                | selectbackground  |
| highlightbackground | selectborderwidth |
| highlightcolor      | selectforeground  |
| highlightthickness  | show              |
| insertbackground    | textvariable      |
|                     | width             |
|                     | xscrollcommand    |