Pythonda Sayılar

Bu bölümle birlikte artık Python öğrenmeye başlıyoruz. Pythonda temeli sağlam atmamız için öncelikle veri tipleri ve veri yapılarını öğrenmeliyiz.İsterseniz sayıları öğrenerek maratonumuza başlayalım.

Bu bölümde şunları öğreneceğiz.

1. Tamsayı(Integer) ve Ondalıklı Sayı(Float) Veri tipleri
2. Basit Matematik İşlemleri
3. Değişken Tanımlama

Tamsayılar (Integer)

Matematikte gördüğümüz tüm sayılar (negatif,pozitif) aslında Python'da bir veri tipidir. Tamsayılar ise ingilizce olarak **Integer** olarak geçmektedir.

Örnek olarak, -1000,34,2,0 gibi sayılar Python'da birer tamsayı(integer) değerleridir.

Ondalıklı Sayılar (Float)

Tamsayılar gibi Ondalıklı sayılar da bizim matematikte gördüğümüz sayı çeşitlerinden bir tanesidir. Ondalıklı sayılar matematikte olduğu gibi, Pythonda da bir veri tipidir.Ondalıklı Sayılar veya diğer adıyla Kayan Sayılar İngilizce olarak **Float** olarak geçmektedir.

Örnek olarak, 3.14, 3.554546, -13.54 gibi sayılar Python'da birer ondalıklı sayı(float) değerleridir.

Şimdi de bu Pythondaki basit matematik işlemlerini öğrenelim.

Basit Matematik Operatörleri

Basit 4 işlemi (Toplama,Çıkarma,Çarpma,Bölme) hepimiz matematikten biliriz. Şimdi bu işlemlerin Pythonda nasıl yapıldığını görmeye çalışalım.

```
In [ ]: # Toplama
        3 + 4

In [3]: #Çıkarma
        5-17

Out[3]: -12

In [4]: #Çarpma
        13 * 4

Out[4]: 52

In [6]: #Bölme
        4/2

Out[6]: 2.0
```

Burada basit işlemlerimizi görmüş olduk. Ancak son işlem kafanızı karıştırmış olabilir. Sonuç neden **float** bir değer olarak karşımıza çıktı? Bunları **Matematik Operatörleri** bölümünde görmek daha doğru olur diye düşünüyorum.

Bu basit işlemleri gördüğümüze göre, şimdi de Pythondaki değişken tanımlama işlemlerimizi öğrenmeye çalışalım.

Değişkenler ve Değişken Tanımlama

Değişkenler bir programlama dilinde olmazsa olmaz bir kavramdır. Değişkenler aslında bir veri tipinden değer tutan birimlerdir.İsterseniz şimdi de Pythonda bir tane değişken oluşturalım.

```
In [8]: # Değişken ismi ve Değişkenin değeri
        i =  10
```

Güzel! Değişkenimizi oluşturduk. Şimdi değişkenin değerini ekrana bastıralım.

```
In [9]: i
Out[9]: 10
```

Ekrana bastırdığımızda değişkenimizin değeri **10** olarak karşımıza çıktı.Peki şu işlemi yaparsak ne olacak ?

```
In [11]: i * i *i
Out[11]: 1000
```

Sonuç, değişkenimizin değerinin 3.kuvveti şeklinde karşımıza çıkmış oldu. Şimdi de değişkenimizin değerini değiştirelim. Bunun için , yeni değerimizi **"="** operatörüyle değişkenimize **atamamız** gerekiyor.

```
In [12]: i = 15
         i
Out[12]: 15
```

Şimdi de 3 tane farklı değişken oluşturmaya çalışalım.

In [1]:

```
In [1]: a = 4
        b = 3
        c = a  + 2*b
```

Peki burada **"c"** değişkeninin değeri ne olacak ? Bu işlemde ilk olarak **=** operatörünün sağ taraftaki işlemi hesaplanıyor ve sonra **c** değişkenine hesaplanan değer atanıyor.

```
In [15]: c
Out[15]: 10
```

İşte bu kadar basit ! Son olarak değişkenlerimize isim verirken dikkat etmemiz gereken noktalardan bahsedelim.

```
        1. Değişken isimleri bir sayı ile başlayamaz.

        2. Değişken ismi kelimelerden oluşuyorsa aralarında boşl
uk olamaz.

        3. :'",<>/?|\()!@#$%^&*~-+ Buradaki semboller değişken i
smi içinde kullanılamaz.(Sadece _ sembolü kullanılabilir)

        4. Pythonda tanımlı anahtar kelimeler değişken ismi olar
ak kullanılamaz.(while, not vs. )
```

```
In [18]: #Hatalı
         i? = 5
```

```
  File "<ipython-input-18-e17f88f28f58>", line 2
    i? = 5
     ^
SyntaxError: invalid syntax
```

```
In [19]: #Doğru
         _i = 5
```

Son olarak öğrendiklerimizle ilgili bir kaç işlem yapalım. Sonuçlara bakmadan işlemlerin sonucunu tahmin etmeye çalışın.

```
In [20]: pi_sayısı = 3.14
         çap = 4
         çevre = pi_sayısı * çap

In [21]: çevre
Out[21]: 12.56
```

Son olarak bir değişkenin değerini artırma işlemlerinde Pythonda pratik bir yöntem bulunmaktadır. Aşağıdaki koda bakalım.

```
In [1]: a = 5
        a = a + 1
        a
Out[1]: 6
```

Buradaki **"a = a + 1"** işlemini Pythonda **"a += 1"** şeklinde yazabiliriz.

```
In [2]: a = 10
        a += 1 # a = a + 1 ile aynı şey.
        a
Out[2]: 11
```

```
In [4]: b = 4

        b -= 1 # b =  b -1
        b
Out[4]: 3
```

```
In [ ]: b = 10
        b *= 3
        b
```

Yorum Satırları

Yorum satırları programlarımıza açıklama olarak eklediğimiz satırlardır. Eğer bir programda yorum satırları kullanılmışsa bu satırlar Python tarafından görülmez ve çalıştırılmaz.

```
In [13]: # Tekli Yorum Satırı
         print("murat")
murat
```

```
In [14]: """
         Çoklu Yorum Satırı

         """
         print("murat")
murat
```

Matematik Operatörleri

Tekrardan hoşgeldiniz.

Python'da ve diğer programlama dillerinde matematik operatörleri oldukça önemlidir. Bu konumuzda, Pythondaki matematik operatörlerini öğreneceğiz.

Toplama İşlemi (+)

Python'da toplama işlemi şu şekilde yapılabilmektedir.

```
In [1]: a = 14
        b = 15
        a + b

Out[1]: 29
```

```
In [4]: i =  3.1
        j  = 4.8
        i+j

Out[4]: 7.9
```

Çıkarma İşlemi (-)

Python'da çıkarma işlemi şu şekilde yapılabilmektedir.

```
In [5]: a =  28
        b = 35
        c = 40
        a-b-c

Out[5]: -47
```

```
In [6]: t = -1
        y = 5
        t-y

Out[6]: -6
```

```
In [7]: k = 3.1
        l = 5.8
        k -l

Out[7]: -2.6999999999999997
```

Çarpma İşlemi (*)

Python'da çarpma işlemi şu şekilde yapılabilmektedir.

```
In [8]: a = 4
        b = 5
        a *b

Out[8]: 20
```

```
In [9]: i = 3.14
        j = 4.5
        i *j

Out[9]: 14.13
```

```
In [10]: a = 3
         b = 4
         c = 5
         a *b *c

Out[10]: 60
```

```
In [19]: a = 3
         b = 3.14
         a *b

Out[19]: 9.42
```

Bölme İşlemi (/)

Python'da bölme işlemi şu şekilde yapılabilmektedir.

```
In [11]: 4 / 2

Out[11]: 2.0
```

```
In [12]: 10 / 3

Out[12]: 3.3333333333333335
```

```
In [13]: 22 / 7

Out[13]: 3.142857142857143
```

Şimdi, burada matematikten farklı olarak bir şey gerçekleşti. **"4 / 2"** işleminin sonucunun **"2"** olarak çıkması gerekirken, sonuç **"2.0"** olarak, yani ondalıklı olarak(float) çıktı. Bu özellik, Python 3'e işlem sonucunun daha kesin bir şekilde sağlanması için konulmuş bir özellik. Bunları ilerde çok daha iyi bir şekilde anlayacağız.
Peki bir sayının başka bir sayıya bölümünden çıkan bölüm sonucunu nasıl bulacağız ? Bunun için de Python3'te şu şekilde bir operatör mevcut.

Kalanı Bulma (%)

Bu operatör de , bir sayının başka bir sayıya bölümünden kalan sonucunu bulmamızı sağlar.

```
In [22]: # 13'ün 4 ile bölümünden kalan 1'dir.
         13 % 4

Out[22]: 1

In [23]: 14 % 2

Out[23]: 0

In [24]: 330 % 111

Out[24]: 108
```

Üs bulma (**)

Bu operatör bir sayının üssünü bulmamızı sağlar. Örnek olarak operatörün solundaki sayının sağdaki sayıya göre üssünü ekrana basar.

```
In [25]: # 4^3
         4 ** 3

Out[25]: 64

In [26]: 2 ** 4

Out[26]: 16

In [27]: 3.14 ** 3

Out[27]: 30.959144000000002
```

Operatörleri beraber kullanma ve İşlem sırası

Bütün bu öğrendiğimiz şeyleri beraber nasıl kullanabiliriz? Bunun için matematikteki işlem sırasını biliyorsak çok rahat edeceğiz, çünkü Pythondaki işlem sırası matematiktekiyle birebir aynıdır. Nedir bu işlem sırası ? Kurallar şunlar ;

1. Parantez içi her zaman önce yapılır.

2. Çarpma ve Bölme her zaman Toplama ve Çıkarma işlemine göre önce yapılır.

3. İşlemler soldan sağa değerlendirilir.

Ancak, buradaki işlemleri ezberlememize hiç gerek yok. Kafamızın karıştığı yerler de işlemleri parantez içine almak, hayat kurtarır :) Şimdi örneklere bakalım.

```
In [32]: 8 + 4 * 3 /2 - 18
Out[32]: -4.0

In [33]: #Kafamız karışıyorsa paranteze alalım.

         8 + ((4 * 3) /2) - 18
Out[33]: -4.0
```

İşte matematik operatörleri tamamıyla bu şekilde !

Şimdi de diğer bir veri tipimiz olan Karakter Dizilerini (**string**) öğrenelim.

Karakter Dizileri (Stringler)

Tekrardan Hoşgeldiniz ! Bu konuda stringleri öğrenmeye çalışacağız.Pythonda bir veri tipi olan Stringler veya Türkçe ismiyle karakter dizileri gerçek hayatta kullandığımız **yazıların** aynısıdır.Bu veri tipi aslında her biri bir karakter olan bir dizidir. Örnek olarak, "ali" stringi sırasıyla a,l,i harflerinden veya karakterlerden oluşmaktadır. Bu konuda stringleri ve stringlerin özelliklerini görmeye çalışacağız.

String Oluşturma

Python'da string oluşturmanın birçok yolu bulunmaktadir.Bunların hangisini kullanacağınız tamamıyla size bağlıdır.İsterseniz string oluşturma işlemlerini görmeye çalışalım.

```
In [1]: #Tek tırnak ile
        'Mustafa Murat Coşkun'

Out[1]: 'Mustafa Murat Coşkun'

In [2]: #Çift Tırnak ile
        "Mustafa Murat Coşkun"

Out[2]: 'Mustafa Murat Coşkun'

In [3]: # 3 tırnak ile
        """Mustafa Murat Coşkun"""

Out[3]: 'Mustafa Murat Coşkun'
```

Burada dikkat etmemiz gereken nokta, eğer bir çift tırnak ile oluşturacaksak, stringin oluşturulması bitirmeyi yine çift tırnak ile yapmalıyız.

```
In [4]: # Hatalı Kod
        "Merhaba'

  File "<ipython-input-4-52c28270f133>", line 2
    "Merhaba'
             ^
SyntaxError: EOL while scanning string literal

In [5]: # Hatalı Kod
        'Merhaba"

  File "<ipython-input-5-688eb17d1742>", line 2
    'Merhaba"
             ^
SyntaxError: EOL while scanning string literal
```

Peki şu şekilde bir stringi Pythonda nasıl tanımlarız: **Murat'ın bugün dersi var.**

```
In [10]: 'Murat'ın bugün dersi var.
  File "<ipython-input-10-c94dae596bd7>", line 1
    'Murat'ın bugün dersi var.
           ^
SyntaxError: invalid syntax
```

Gördüğümüz gibi kodumuz sıkıntı çıkarttı. Çünkü karşılaştığı ilk ' işaretini bitirme işareti olarak algıladı. Ancak şöyle bir kullanım sıkıntı yaratmayacaktır.

```
In [11]: "Murat'ın bugün dersi var"
Out[11]: "Murat'ın bugün dersi var"
```

Şimdi de bir tane string veri tipinde değişken oluşturalım. Stringler de bir veri tipi oldukları için bunlardan değişken oluşturup kullanabiliriz.

```
In [9]: a = "Merhaba"
        a
Out[9]: 'Merhaba'

In [13]: naber = "Naber iyi misin ?"
         naber
Out[13]: 'Naber iyi misin ?'
```

String Indeksleme ve Parçalama

Stringler birer karakter dizileri oldukları için her bir karakterin aslında string içinde bir yeri vardır. Örnek olarak **"ali"** stringinde a,l ve i karakterlerinin yerleri **indeks** olarak adlandırılır.Pythonda ve genellikle çoğu programlama dilinde (Octave hariç) stringlerin indekslenmesi **"0"** dan başlamaktadır. Şimdi isterseniz bir string içindeki karakterlere **indeks** yoluyla nasıl ulaşacağımıza bakalım.

```
In [14]: # 0. elemana  ulaşalım . Bunun için [] operatörünü kullanacağız.
         a = "ali"
         a[0]
Out[14]: 'a'

In [15]: # 1.eleman
         a[1]
Out[15]: 'l'

In [16]: a[2]
Out[16]: 'i'
```

Pythonda stringler baştan olduğu gibi sondan da indekslenebilirler. Sondan başlayarak -1,-2 ... şeklinde indekslenirler

```
In [18]: # Sondaki eleman "-1" eleman
         a = "murat"
         a[-1]
Out[18]: 't'

In [19]: a[-2]
Out[19]: 'a'

In [20]: a[-3]
Out[20]: 'r'

In [21]: a[-4]
Out[21]: 'u'

In [22]: a[-5]
Out[22]: 'm'
```

Peki uzun bir string'in sadece belirli bir kısmını elde etmek için ne yapacağız ? Bunun için **indeksleri**, **:** ve **[ ]** işaretini kullanacağız. Formülümüz şu şekilde ;

[başlama indeksi : bitiş indeksi : atlama değeri]

İsterseniz örneklerimize bakalım.

```
In [23]: a = "Python Programlama Dili"

In [25]: # 4. indeksten başla 10.indekse kadar(dahil değil) al.
         a[4:10]
Out[25]: 'on Pro'

In [26]: # Başlangıç değeri belirtilmemişse en baştan başlayarak alır.
         a[:10]
Out[26]: 'Python Pro'

In [28]: # Bitiş değeri belirtilmemişse en sonuna kadar alır.
         a[4:]
Out[28]: 'on Programlama Dili'

In [29]: # İki değer de belirtilmemişse tüm stringi al.
         a[:]
Out[29]: 'Python Programlama Dili'
```

```
In [30]: #Son karaktere kadar al.
         a[:-1]
Out[30]: 'Python Programlama Dil'

In [32]: # Baştan sona 2 değer atlaya atlaya stringi al.
         a[::2]
Out[32]: 'Pto rgalm ii'

In [33]: # 4.indeksten 12'nci indekse 3'er atlayarak stringi al.
         a[4:12:3]
Out[33]: 'oPg'

In [34]: # Baştan sona -1 atlayarak stringi al. (String'i ters çevirme)
         a[::-1]
Out[34]: 'iliD amalmargorP nohtyP'
```

String Özellikleri

Bir string'in uzunluğunu nasıl buluruz ? Bunun için Python'da *len()* fonksiyonu bulunmaktadır.

```
In [35]: # len() fonksiyonunu kullanma.
         a = "Python Programlama Dili"
         len(a)
Out[35]: 23
```

Peki, Pythonda stringler toplanabiliyor mu ? Python'da bunu yapmak da mümkündür.

```
In [39]: # Stringleri toplayalım yani birbirine ekleyelim.
         a = "Python "
         b = "Programlama "
         c = "Dili"
         a + b + c
Out[39]: 'Python Programlama Dili'

In [3]: a = "Mustafa "
        a = a + "Murat Coşkun" # Burada stringleri birleştirerek yine a değişkenine atıyoruz.
        a
Out[3]: 'Mustafa Murat Coşkun'
```

Bir string ile bir sayıyı da çarpabiliriz.

```
In [41]: # Python * 3 aslında Python + Python + Python işlemine eşdeğerdir.
         "Python" * 3
Out[41]: 'PythonPythonPython'
```

Stringler konusu şimdilik bu kadar ! İleriki konularımızda stringlerimize tekrardan döneceğiz ve güzel özelliklerine bakma imkanına sahip olacağız.

Print Fonksiyonu

Bu bölümde ekrana veri tiplerini yazdırmak için kullandığımız *print()* fonksiyonunu ve formatlama yöntemlerini öğreneceğiz.

Print() Fonksiyonu

Kodlarımızı dosyalara yazdığımızda, eğer ekrana bir değer bastırmak istersek *print* fonksiyonunu kullanırız. Kullanımı oldukça basittir ve değişik özelliklere sahiptir. Örneklerimize bakalım.

```
In [1]: print(35)
```

```
35
```

```
In [2]: print(3.14)
```

```
3.14
```

```
In [3]: a = 4
        b = 15
        print(a+b)
```

```
19
```

```
In [4]: print("Mustafa Murat Coşkun")
```

```
Mustafa Murat Coşkun
```

```
In [5]: print("Murat'ın bugün dersi var.")
```

```
Murat'ın bugün dersi var.
```

Buradaki işlemlerde gördüğümüz gibi biz print fonksiyonunun içine bastırmak istediğimiz değeri veriyoruz ve bu fonksiyon da ekrana değerimizi bastırıyor. Peki aynı satırda birkaç değer bastırmak istersek ne yapıyoruz? Bunun için değerlerimizin arasına **,** karakterini atıyoruz.

```
In [6]: print("Murat",12,545,66767,3.56)
```

```
Murat 12 545 66767 3.56
```

```
In [7]: print("Mustafa","Murat","Coşkun")
```

```
Mustafa Murat Coşkun
```

type() fonksiyonu

print() fonksiyonundan bahsetmişken *type()* fonksiyonunu öğrenmekte fayda var. *type()* fonksiyonu içine gönderilen değerin hangi veri tipinden olduğunu söyler.

In [20]:
```
# Integer (Tamsayı) türü
a = 65
print(type(a))
```

```
<class 'int'>
```

In [21]:
```
# Float (Ondalıklı Sayı) türü
a = 5.87
print(type(a))
```

```
<class 'float'>
```

In [22]:
```
# String (Karakter Dizisi) türü
a = "Murat"
print(type(a))
```

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

Kullanıcıdan Girdi Alma - input() Fonksiyonu

Programlama yaparken kullanıcıdan girdi almak oldukça önemlidir. Pythonda kullanıcıdan girdi almamızı sağlayan *input()* fonksiyonu bulunmaktadır.

input() fonksiyonu kullanımı

input() fonksiyonu şu şekilde kullanılabilir.

```
In [3]: input() # Çalıtırdığımız zaman input fonksiyonu bizden bir girdi bekler.
        25
Out[3]: '25'
```

Eğer istersek fonksiyonun içine bir değer de gönderebiliriz.

```
In [4]: input("Lütfen bir sayı giriniz:")
        Lütfen bir sayı giriniz:25
Out[4]: '25'
```

Kullanıcıdan aldığımız değeri input() fonksiyonu yoluyla şu şekilde elde edebiliriz.

```
In [1]: a = input("Lütfen bir sayı giriniz:") # Kullanıcıdan aldığımız değer a değişkenine eşit olacak.
        print("Kullanıcının girdiği değer:",a)
        Lütfen bir sayı giriniz:25
        Kullanıcının girdiği değer: 25
```

Kullanıcının girdiği değer **input** fonksiyonundan string olarak bize döner.

```
In [2]: a = input("Lütfen bir sayı giriniz:")
        print(type(a))
        Lütfen bir sayı giriniz:26
        <class 'str'>
```

Eğer biz bir sayı ile işlem yapacaksak kullanıcıdan aldığımız değere (stringi) *float* ya da *int* fonksiyonuyla **veri tipi** dönüştürme işlemi yapmamız gerekir. Çünkü aşağıdaki gibi bir program yanlış çalışacaktır.

```
In [3]: # Hatalı Çalışma
        a = input("Lütfen bir sayı giriniz:")
        print(a * 3) # Girdiğimiz değer 5 ise sonucu 15 bekleriz. Ancak sonuç 555 şeklinde ortaya çıkar.
        Lütfen bir sayı giriniz:5
        555

In [4]: # Doğru Çalışma
        a = int(input("Lütfen bir sayı giriniz:")) # Veri tipi dönüşümü
        print(a * 3)
        Lütfen bir sayı giriniz:5
        15

In [5]: # Hatalı Çalışma
        a = input("Lütfen bir sayı giriniz:")
        print(a * 3) # Girdiğimiz değer 5.4 ise sonucu 16.2 bekleriz. Ancak sonuç 5.45.45.4 şeklinde ortaya çıkar.
        Lütfen bir sayı giriniz:5.4
        5.45.45.4

In [6]: # Doğru Çalışma
        a = float(input("Lütfen bir sayı giriniz:"))
        print(a * 3) # Girdiğimiz değer 5.4 ise sonucu 16.2 bekleriz. Ancak sonuç 5.45.45.4 şeklinde ortaya çıkar.
```

Bir tane örnek program yazalım.

In [1]:
```
a = int(input("Birinci Sayı:"))
b = int(input("İkinci Sayı:"))
c = int(input("Üçüncü Sayı:"))

print("Toplamları:",a+b+c)
```

```
Birinci Sayı:12
İkinci Sayı:24
Üçüncü Sayı:54
Toplamları: 90
```

Listeler

Şimdi de yeni bir veritipimiz olan **listeleri** öğrenmeye çalışalım. Listeler yapıları gereği stringlere oldukça benzerler ve kullanıldıkları yerler de çok yararlı olan bir veritipidir. Tıpkı stringler gibi ,indekslenirler,parçalanırlar ve üzerinde değişik işlemler yapabildiğimiz metodlar bulunur. Ancak listelerin stringlerden önemli farkları da bulunmaktadır. Stringler konusundan bildiğimiz kadarıyla stringler değiştirilemez bir veri tipidir. Ancak, listelerimiz değiştirilebilir bir veritipidir.

Bir listede her veritipinden elemanı saklayabiliriz. Bu anlamda sıralı bir diziye benzer. Peki bu konuda ne öğreneceğiz?

```
1.Liste oluşturma

2.Indeksleme ve Parçalama

3.Temel Liste Metodları ve İşlemleri
```

Liste Oluşturma

Listeler bir [ ] köşeli parantez içine veriler veya değerler konarak oluşturulabilir.

In [1]:
```
# liste değişkeni. Değişik veri tiplerinden değerleri saklayabiliyoruz.
liste =  [3,4,5,6,"Elma",3.14,5.324]
liste
```

Out[1]:
```
[3, 4, 5, 6, 'Elma', 3.14, 5.324]
```

In [2]:
```
liste2 = [3,4,5,6,7,8,9]
liste2
```

Out[2]:
```
[3, 4, 5, 6, 7, 8, 9]
```

In [3]:
```
# Boş liste
bos_liste = []
bos_liste
```

Out[3]:
```
[]
```

In [4]:
```
# Boş liste . list() fonksiyonuyla da oluşturulabilir.
bos_liste = list()
bos_liste
```

Out[4]:
```
[]
```

```
In [5]: # len fonksiyonu listeler üzerinde de kullanılabilir.
        liste3 = [3,4,5,6,6,7,8,9,0,0,0]
        len(liste3)
Out[5]: 11
```

Bir string *list()* fonksiyonu yardımıyla listeye dönüştürülebilir.

```
In [6]: s =  "Merhaba"
        lst =  list(s)
        lst
Out[6]: ['M', 'e', 'r', 'h', 'a', 'b', 'a']
```

Listeleri Indeksleme ve Parçalama

Listeleri indeksleme ve parçalama stringlerle birebir olarak aynıdır.

```
In [7]: liste = [3,4,5,6,7,8,9,10]
        liste
Out[7]: [3, 4, 5, 6, 7, 8, 9, 10]

In [8]: # 0. eleman
        liste[0]
Out[8]: 3

In [9]: # 4. eleman
        liste[4]
Out[9]: 7

In [10]: # Sonuncu Eleman
         liste[len(liste)-1]
Out[10]: 10

In [11]: # Sonuncu Eleman
         liste[-1]
Out[11]: 10
```

```
In [13]: # Baştan 4. indekse kadar (dahil değil)
         liste[:4]
Out[13]: [3, 4, 5, 6]

In [14]: # 1.indeksten 5.indekse kadar
         liste[1:5]
Out[14]: [4, 5, 6, 7]

In [15]: liste[5:]
Out[15]: [8, 9, 10]

In [16]: liste[::2]
Out[16]: [3, 5, 7, 9]

In [17]: liste[::-1]
Out[17]: [10, 9, 8, 7, 6, 5, 4, 3]
```

Temel Liste Metodları ve İşlemleri

Bu kısımda da listelerde yapabileceğimiz temel işlemleri ve bazı temel metodları öğreneceğiz. Listelerin daha bir çok metodunu kursun ileriki kısımlarında görüyor olacağız.

Bir listeyi birbirine toplama işlemini stringlerdeki gibi yapabiliriz.

```
In [18]: liste1 =  [1,2,3,4,5]
         liste2 =  [6,7,8,9,10]
         liste1 + liste2
Out[18]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Bir listeye bir tane eleman eklemek içinse aşağıdaki işlemi uygulayabiliriz.

```
In [19]: liste = [1,2,3,4]
         liste =  liste + ["Murat"]

In [20]: liste
Out[20]: [1, 2, 3, 4, 'Murat']

In [21]: # Listeler direk olarak değiştirilebilirler.
         liste[0] = 10

In [22]: liste
Out[22]: [10, 2, 3, 4, 'Murat']
```

```
In [23]: # Şöyle bir kullanım da mümkündür.
         liste[:2] = [40,50]

In [24]: liste
Out[24]: [40, 50, 3, 4, 'Murat']
```

Bir listeyi bir tamsayıyla da çarpabiliriz.

```
In [25]: liste = [1,2,3,4,5]
         liste * 3
Out[25]: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

In [26]: liste # Ama gördüğümüz gibi listemiz değişmiyor.
Out[26]: [1, 2, 3, 4, 5]

In [27]: liste = liste * 3

In [28]: liste # Eşitleme yaptığımız için liste değişti.
Out[28]: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

Şimdi de isterseniz temel bazı liste metodlarına bakalım. Listeler, diğer programlama dillerindeki **arraylere** göre oldukça esnektir. Belli bir boyutları yoktur ve ekleme, çıkarma yapmak oldukça kolaydır.

append metodu

append metodu, verdiğimiz değeri listeye eklememizi sağlar.

```
In [29]: liste = [3,4,5,6]
         liste.append(7)
         liste
Out[29]: [3, 4, 5, 6, 7]

In [30]: liste.append("Murat")

In [31]: liste
Out[31]: [3, 4, 5, 6, 7, 'Murat']
```

Listeye eleman eklemenin ikinci bir yolu + operatörünü kulanmaktır.

+ operatörünü kullanacaksak yeni elemanında bir liste elemanına dönüşmesini sağlamak için, yeni elemanı [ ] köşeli parantez içine yazmayı unutmayalım.

In [1]:
```
#listeye yeni eleman eklemeyi + operatörü ile de yapabiliriz.
liste = [3,4,5,6]
liste=liste + [7]  # 7 elemanının liste elemanı olabilmesi için
liste              # [ ] köşeli parantez kullanmayı unutmayalım
```

Out[1]:
```
[3, 4, 5, 6, 7]
```

In [4]:
```
liste=liste+["Sosyal Bilimler Lisesi"]
```

In [5]:
```
liste
```

Out[5]:
```
[3, 4, 5, 6, 7, 'Murat', 'Sosyal Bilimler Lisesi']
```

Mantıksal Değerler ve Karşılaştırma Operatörleri

Bu konuda, koşullu durumları incelemeden önce bilmemiz gereken mantıksal değerleri ve karşılaştırma operatörlerini öğrenmeye çalışacağız.

Mantıksal Değerler (Boolean)

Mantıksal değerler ya da ingilizce ismiyle *boolean* değerler aslında Pythonda bir veri tipidir ve iki değere sahiplerdir: **True ve False**. Şimdi bu değerlerden değişkenler oluşturalım.

Karşılaştırma Operatörleri

Operatör	Görevi	Örnek
==	İki değer birbirine eşitse True, değilse False değer döner.	2 == 2 (True) , 2 == 3 (False)
!=	İki değer birbirine eşit değilse True, diğer durumda False döner.	2 != 2 (False), 2 != 3 (True)
>	Soldaki değer sağdaki değerden büyükse True, değilse False döner.	3 > 2 (True), 2 > 3 (False)
<	Soldaki değer sağdaki değerden küçükse True, değilse False döner.	2 < 3 (True) , 3 < 2 (False)
>=	Soldaki değer sağdaki değerden büyükse veya sağdaki değere eşitse True, değilse False döner.	3 >= 2 (True), 3 >= 3 (True) , 2 >= 3 (False)
<=	Soldaki değer sağdaki değerden küçükse veya sağdaki değere eşitse True, değilse False döner.	3 <= 2 (False), 3 <= 3 (True) , 2 <= 3 (True)

Örneklere bakalım.

```
In [13]: "Mehmet" == "Mehmet"
Out[13]: True

In [14]: "Mehmet" == "Murat"
Out[14]: False

In [15]: "Mehmet" != "Murat"
Out[15]: True

In [16]: "Oğuz" < "Murat" # Alfabetik olarak bakar.
Out[16]: False

In [17]: 2 < 3
Out[17]: True
```

Buradaki karşılaştırma operatörlerini koşullu durumlarda koşulları kontrol etmek için kullanacağız. Şimdi mantıksal bağlaçlar konusuna geçebiliriz.

Mantıksal Bağlaçlar

Mantıksal bağlaçlar daha çok karşılaştırma işlemini kontrol ettiğimiz zamanlarda kullanılır. Bu konuda bunları öğrenmeye çalışacağız.

and Operatörü

Bu mantıksal bağlaç, bütün karşılaştırma işlemlerinin sonucunun **True** olmasına bakar. Bağlanan karşılaştırma işlemlerinin hepsinin kendi içinde sonucu **True** ise genel sonuç **True** , diğer durumlarda ise sonuç **False** çıkar. Kullanımı şu şekildedir.

```
In [1]: 1 < 2 and "Murat" == "Murat"
Out[1]: True

In [2]: 2 > 3 and "Murat" == "Murat"
Out[2]: False

In [3]: 2 == 2 and 3.14 < 2.54 and "Elma" != "Armut"
Out[3]: False
```

İşlemlerin birinin bile sonucu **False** ise genel işlemin sonucu **False** çıkmaktadır.

or Operatörü

Bu mantıksal bağlaç, bütün karşılaştırma işlemlerinin sonuçlarından **en az birinin True** olmasına bakar. Bağlanan karşılaştırma işlemlerinin **en az birinin True** olmasında genel sonuç **True** , diğer durumlarda ise sonuç **False** çıkar. Kullanımı şu şekildedir.

```
In [4]: 1 < 2 or "Murat" != "Murat"
Out[4]: True

In [5]: 2 > 3 or "Murat" != "Murat"
Out[5]: False

In [6]: 2 > 3 or "Murat" != "Murat" or 3.14 < 4.32
Out[6]: True
```

not operatörü

not operatörü aslında bir mantıksal bağlaç değildir. Bu operatör sadece bir mantıksal değeri veya karşılaştırma işlemininin tam tersi sonuca çevirir. Yani, not operatörü **True** olan bir sonucu **False** , **False** olan bir sonucu **True** sonucuna çevirir. Kullanımı şu şekildedir.

```
In [7]: not 2 == 2
Out[7]: False

In [9]: not "Python" == "Php"
Out[9]: True
```

Operatörleri Beraber Kullanma

Burada gördüğümüz 3 operatörü beraber kullanmak karmaşıklığa yol açacağı için, *parantez* kullanabiliriz.

```
In [20]: not (2.14 > 3.49 or ( 2 != 2 and "Murat" == "Murat"))
Out[20]: True

In [22]: "Araba" < "Zula" and ( "Bebek" < "Çoçuk" or (not 14 ))
Out[22]: True
```

Koşullu Durumlar - if-else koşullu durumları

Şimdiye kadar Pythondaki temel veritiplerini öğrendik ve gerekli temelleri aldık. Artık bundan sonra çok daha eğlenceli konular göreceğiz ve elle tutulur programlar yazmaya başlayacağız. Bu konuda ilk olarak Python programlarının çalışma mantığını göreceğiz ve daha sonrasında koşullu durumları anlamaya çalışacağız.

Python Programlarının çalışma mantığı

Python programları çalışmaya başladığı zaman kodlarımız yukardan başlayarak teker teker çalıştırılır ve çalıştıracak kod kalmayınca programımız sona erer. Şöyle bir örneği düşünelim;

```
In [1]: a = 2
        b = 3
        c = 4
        print(a+b+c)
```

```
9
```

Yandaki basit kodda program teker teker her bir satırı ve ifadeyi çalıştırır ve çalıştıracak kod kalmayınca program sona erer.Ancak Python'da her program bu kadar basit olmayabilir. Çoğu zaman Python programlarımız belirli bloklardan oluşur ve bu bloklar her zaman çalışmak zorunda olmaz.Peki bu bloklar nasıl tanımlanır ? Pythonda bir blok tanımlama işlemi **Girintiler** sayesinde olmaktadır. Örnek olması açısından, Pythonda bloklar şu şekilde oluşabilir.

```
In [1]: a = 2 # Blok 1 'e ait kod

        if (a == 2):
            print(a) # Blok 2'ye ait kod
        print("Merhaba") # Blok 1 ' ait kod
```

```
2
Merhaba
```

Dikkat ederseniz burada daha önce görmediğimiz bir şey yaptık ve **if** in bulunduğu satırdan sonraki **print** işlemini bir tab kadar girintili yazdık. Burada gördüğümüz gibi, **girintiler(tab)** Pythonda bir blok oluşturmak için kullanılıyor ve her bloğunun çalıştırılması gerekmiyor. Mesela yukarıda gördüğümüz kodda **2 print** işlemi de çalıştı. Ancak kodumuzu şu şekilde yazsaydık, ilk **print** işlemi çalışmayacaktı.

```
In [2]: a = 2 # Blok 1 'e ait kod

        if (a == 3):
            print(a) # Blok 2'ye ait kod
        print("Merhaba") # Blok 1 ' ait kod
```

```
Merhaba
```

Buradaki blok tanımlama işlemlerimiz bundan sonra sürekli karşımıza çıkacak. Eğer henüz anlamadıysanız zamanla anladığınızı göreceksiniz.

Koşullu Durumlar

Artık Pythonda bizi bir ileri seviyeye taşıyacak koşullu durumları öğrenmenin vakti geldi.

Koşullu durumlar aslında günlük yaşamda sürekli karşılaştığımız durumlardır. Örneğin havanın yağmurlu olma koşuluna göre şemsiyemizi alırız veya uykumuzun gelme koşuluna göre uyuruz. Aslında programlamada da birçok koşullu durumla karşılaşırız. Örneğin , belli koşullara göre belli işlemleri yaparız , belli koşullara göre yapmayız. İşte bunlar koşullu durumların temeli oluşturur. İsterseniz koşullu durumları yazmaya **if** blokları ile başlayalım.

if Bloğu

if bloğu programımızın içinde herhangi bir yerde belli bir koşulu kontrol edecek isek kullanılan bloklardır.Yazımı şu şekildedir;

```
if (koşul):

    # if bloğu - Koşul sağlanınca (True) çalışır. Bu hiza
daki her işlem bu if bloğuna ait.

    # if bloğu - Girintiyle oluşturulur.

    Yapılacak İşlemler
```

if bloğu *eğer koşul sağlanırsa anlamı taşır*. Eğer if kalıbındaki koşul sağlanırsa (True) if bloğu çalıştırılır, koşul sağlanmazsa (False) if bloğu çalıştırılmaz.

Hemen bir örnek ile koşullu durumları anlamaya çalışalım.

In [10]:
```
# Negatif mi değil mi ?
sayı = int (input("Sayıyı giriniz:"))

if (sayı < 0):
    print("Negatif Sayı")
```

```
Sayıyı giriniz:-3
Negatif Sayı
```

Başka bir örnek yapalım.

In [7]:
```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
```

In [9]:
```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
```

```
Yaşınızı giriniz:25
```

Ancak buralarda bir eksiklik var. Örneğin , ** 18 yaş kontrolü ** örneğinde eğer *yaş* 18'e eşit ve 18'den büyükse ekrana herhangi bir şey basamıyoruz. Şöyle bir şey acaba istediğimizi gerçekleştirecek mi ?

In [11]:
```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:25
Mekana hoşgeldiniz.
```

Burada istediğimiz sağlanmış gibi görünüyor. Ancak durum öyle değil. Kodumuzu bir kere daha çalıştıralım ve yaşımızı 17 olarak girelim.

```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
Mekana hoşgeldiniz.
```

Burada istediğimizi elde edemedik çünkü 2. print işlemi herhangi bir koşula bağlı değil. O yüzden 2.print işlemi her durumda çalışıyor. Peki **if koşulu** sağlanmadığında belli bir işlemin çalışmasını nasıl sağlayacağız ? Bunun için sıradaki kalıbımızı öğrenelim.

else Bloğu

else blokları **if koşulu ** sağlanmadığı zaman (False) çalışan bloklardır. Kullanımı şu şekildedir;

```
else:

    # else bloğu - Yukarısındaki herhangi bir if bloğu (v
eya ilerde göreceğimiz elif bloğu) çalışmadığı

    # zaman çalışır.

    # else bloğu - Girintiyle oluşturulur.

        Yapılacak İşlemler
```

In [13]:
```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
else:
    # else bloğu -  if koşulu sağlanmazsa çalışacak.
    print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:25
Mekana hoşgeldiniz.
```

In [14]:
```
# 18 yaş kontrolü
yaş = int(input("Yaşınızı giriniz:"))

if (yaş < 18):
    # if bloğu -  Girinti ile sağlanıyor.
    print("Bu mekana giremezsiniz.")
else:
    # else bloğu -  if koşulu sağlanmazsa çalışacak.
    print("Mekana hoşgeldiniz.")
```

```
Yaşınızı giriniz:17
Bu mekana giremezsiniz.
```

İşte koşullu durumların aslında mantığı tamamıyla bu kadar ! Konuyu bitirmeden önce aşağıdaki kodu çalıştıralım.

Koşullu Durumlar - if - elif - else koşullu durumları

Önceki konumuzda koşullu durumlardaki if-else kalıplarımızı öğrendik. Bu bölümde de if-elif-else kalıplarını öğrenmeye çalışacağız.

if-elif-else Blokları

Önceki konumuzda koşullu durumlarımızla sadece tek bir koşulu kontrol edebiliyorduk. Ancak programlamada bir durum bir çok koşula bağlı olabilir. Örneğin bir hesap makinesi programı yazdığımızda kullanıcının girdiği işlemlere göre koşullarımızı belirleyebiliriz. Bu tür durumlar için if-elif-else kalıplarını kullanırız. Kullanımı şu şekildedir;

```
if koşul:

    Yapılacak İşlemler

elif başka bir koşul:

    Yapılacak İşlemler

elif başka bir koşul:

    Yapılacak İşlemler


     //

     //

else:

    Yapılacak İşlemler
```

Programlarımızda, Kaç tane koşulumuz var ise o kadar elif bloğu oluşturabiliriz. Ayrıca **else** in yazılması zorunlu değildir. if - elif - else kalıplarında, hangi koşul sağlanırsa program o bloğu çalıştırır ve if-elif-blokları sona erer. Şimdi isterseniz kullanıcıya işlem seçtirdiğimiz bir programla , bu kalıbı öğrenmeye başlayalım.

In [1]:
```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:1
1. işlem seçildi.
```

In [2]:
```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:3
3. işlem seçildi.
```

In [3]:
```
işlem =  int(input("İşlem seçiniz:")) # 3 tane işlemimiz olsun.

if işlem == 1:
    print("1. işlem seçildi.")
elif işlem == 2:
    print("2. işlem seçildi.")
elif işlem == 3:
    print("3. işlem seçildi.")
else:
    print("Geçersiz İşlem!")
```

```
İşlem seçiniz:4
Geçersiz İşlem!
```

Programlarda **else** kalıbının kullanılmasına gerek yoktur.Buradaki kodda biz diğer durumlar için sadece opsiyonel bir **else** bloğu koyduk. Kodumuz **else** bloğu olmadan da çalışabilecektir. Ancak bu durumda yanlış bir işlem girilirse ekrana herhangi bir şeyin yazılmadığını göreceğiz.
Yani, **else** bloğu kullanmak tamamen size bağlı.

if-if-if Blokları

Bu blokları öğrenmeden önce isterseniz öğrendiğimiz bilgilerle, bir eski sistem ile karne notu hesaplama programı yapalım. Daha sonra bu kalıpları anlamaya çalışalım.

In [2]:
```
note = int (input("Notunuzu giriniz:"))

if note >= 85:
    print("PEKİYİ")
elif note >= 70:
    print("İYİ")
elif note >= 55:
    print("ORTA")
elif note >= 45:
    print("GEÇER")

else:
    print("Zayıf Dersten Kaldınız")
```

```
Notunuzu giriniz:80
İYİ
```

Burada eğer herhangi bir bloğumuz koşulu sağlarsa **print** işlemi gerçekleşecek ve programımız sonlanacaktır. Ancak acaba **elif** bloklarını **if** bloklarına çevirirsek programımız nasıl çalışacak ? Hemen bakalım.

In [3]:
```
note = int(input("Notunuzu giriniz:"))

if note >= 85:
    print("PEKİYİ")
if note >= 70:
    print("İYİ")
if note >= 55:
    print("ORTA")
if note >= 45:
    print("GEÇER")
else:
    print("Zayıf Dersten Kaldınız")
```

```
Notunuzu giriniz:80
İYİ
ORTA
GEÇER
```

Burada gördüğümüz gibi programımız beklenmedik (istenmedik) bir şekilde çalıştı. Girdiğimiz 80 notu ikinci **if** blokunda sağlandığı halde, üçüncü (note>=55) ve dördüncü (note>=45) if bloklarını da kontrol ederek bu şartlarında sağlandığını gördü. Çünkü Pythonda programlar **her zaman** bütün **if** bloklarını kontrol eder ve koşullar doğruysa bu blokları çalıştırır. İşte böyle not hesaplama gibi programlarda **elif** kullanmamızın sebebi budur.

Koşullu durumlarımız şimdilik bu kadar ! Kurs boyunca programlarımızda birçok yerde koşullu durumları kullanacağız. Bu kısım sonundaki alıştırma videolarını izlerseniz ve ödevleri yaparsanız iyi yol kat etmiş olacaksınız.

Döngü Yapılarını Kullanma

Şimdiye kadar yazdığımız programlarda yazdığımız programlar bir defa çalışıyor ve sona eriyordu. Ancak biz çoğu zaman programlarımızın belli koşullarda çalışmasını sürekli devam ettirmesini ve işlemlerini tekrar etmesini isteriz. İşte bunları yapmamızı sağlayan yapılara **döngü** diyoruz.

Döngüler bütün programlama dillerinde bulunan ve belli koşullarda işlemlerini sürekli tekrar eden yapılardır. İsterseniz gerçek hayattaki programlara bakarak döngü mantığını anlamaya çalışalım.

Örneğin , siz ATM makinesine gidip kartınızı yerleştiriyorsunuz ve program başlıyor. Para Çekme, Para Yatırma , Vergileri Ödeme gibi işlemleri tekrar tekrar gerçekleştiriyorsunuz. Programın sona ermesi ise Kart İade seçeneği ile gerçekleşiyor. Yani siz Kart İade tuşuna basmadığınız sürece ATM makinesi çalışmaya devam ediyor. Buna bakarak ,aslında ATM makinesi döngü yapılarını kullanıyor diyebiliriz.

Başka bir örnek düşünelim. Örneğin siz bir siteye login olma işlemi gerçekleştiriyorsunuz. Biz kullanıcı adı ve parolayı yanlış girdiğimiz sürece program sürekli bize kullanıcı adı ve parola soruyor. Programın sona ermesi ise biz kullanıcı adı ve parolayı doğru girdiğimizde gerçekleşiyor. Yine burada da siz döngü yapılarının kullanıldığını düşünebilirsiniz.

Biz de artık bu bölümle birlikte Pythondaki while ve for döngülerini kullanarak programlarımızı daha efektif bir şekilde yazabileceğiz.

in Operatörü

Pythondaki *in* operatörü , bir elemanın başka bir listede,demette veya stringte (karakter dizileri) bulunup bulunmadığını kontrol eder. Kullanımı şu şekildedir;

```
In [1]: "a" in "merhaba"
Out[1]: True

In [4]: "mer" in "merhaba"
Out[4]: True

In [5]: "t" in "merhaba"
Out[5]: False

In [6]: 4 in [1,2,3,4]
Out[6]: True
```

```
In [7]: 10 in [1,2,3,4]
Out[7]: False

In [8]: 4 in (1,2,3)
Out[8]: False
```

range() Fonksiyonu

Pythondaki bu hazır fonksiyon bizim verdiğimiz değerlere göre *range* isimli bir yapı oluşturur ve bu yapı listelere oldukça benzer. Bu yapı başlangıç, bitiş ve opsiyonel olarak artırma değeri alarak listelere benzeyen bir sayı dizisi oluşturur. Kullanımlarını öğrenmeye başlayalım.

```
In [11]: range(0,20) # 0'dan 20' a kadar (dahil değil) sayı dizisi oluşturur.
Out[11]: range(0, 20)

In [13]: print(*range(0,20)) # Yazdırmak için başına "*" koymamız gerekiyor.
         0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

In [17]: liste = list(range(0,20)) # list fonksiyonuyla listeye dönüştürebilir.

In [16]: liste
Out[16]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [18]: print(*range(5,10))
         5 6 7 8 9

In [20]: print(*range(15)) # Başlangıç değeri vermediğimiz 0'dan başlar
         0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

In [22]: print(*range(5,20,2)) # 5'ten 20'ye kadar olan sayıları 2 atlayarak oluşturur.
         5 7 9 11 13 15 17 19

In [23]: print(*range(5,100,5)) # 5'ten 100'e kadar olan ve 5 ile bölünebilen sayılar
         5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95

In [28]: print(*range(20,0)) # 20'den geri gelen sayıları oluşturmaz.


In [29]: print(*range(20,0,-1)) # 20'den geri gelen sayıları oluşturur.
         20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Şimdi de, *range* fonksiyonu ile oluşturduğumuz yapının üzerinde for döngüsü ile gezinelim.

for Döngüsü

for Döngüsü , listelerin ,demetlerin, stringlerin ve hatta sözlüklerin üzerinde dolaşmamızı sağlayan bir döngü türüdür. Yapısı şu şekildedir.

```
for eleman in veri_yapısı(liste,demet vs):

    Yapılacak İşlemler
```

Bu yapı bize şunu söyler;

```
eleman değişkeni her döngünün başında listenin,demetin v
s. her bir elemanına eşit olacak ve her döngüde

bu elemanla işlem yapılacak.
```

for döngüsünü daha iyi anlamak için örneklerimize bakalım.

Listeler Üzerinde Gezinmek

```
In [2]: # Liste elemanları yazdırma
liste = [1,2,3,4,5,6,7]
for eleman in liste:
    print(eleman)
```

```
1
2
3
4
5
6
7
```

```
In [2]: # Liste elemanlarını toplama
liste = [1,2,3,4,5,6,7]
toplam = 0
for eleman in liste:
    toplam += eleman
print("Toplam",toplam)
```

```
Toplam 28
```

```
In [3]: # Çift elemanları bastırma
liste = [1,2,3,4,5,6,7,8,9]

for eleman in liste:
    if eleman % 2 == 0:
        print(eleman)
```

```
2
4
6
8
```

Karakter Dizileri Üzerinde Gezinmek (Stringler)

In [4]:
```
s =  "Python"
for karakter in s:
    print(karakter)
```

```
P
y
t
h
o
n
```

In [5]:
```
# Her bir karakterleri 3 ile çarpma
s = "Python"

for karakter in s:
    print(karakter * 3)
```

```
PPP
yyy
ttt
hhh
ooo
nnn
```

While Döngüleri

Bu bölümde *while* döngülerinin yapısını ve nasıl kullanılacağını öğrenmeye çalışacağız.

while döngüleri belli bir koşul sağlandığı sürece bloğundaki işlemleri gerçekleştirmeye devam eder. *while* döngülerinin sona ermesi için koşul durumunun bir süre sonra **False** olması gereklidir.Yapısı şu şekildedir;

```
while (koşul):

    İşlem1

    İşlem2

    İşlem3

      //

      //
```

while döngülerini daha iyi anlamak için örneklerimize bakalım.

In [1]:
```
# Döngüde i değerlerini ekrana yazdırma

i = 0

while (i < 10):
    print("i'nin değeri",i)
    i += 1 # Koşulun bir süre sonra False olması için gerekli - Unutmayalım
```

```
i'nin değeri 0
i'nin değeri 1
i'nin değeri 2
i'nin değeri 3
i'nin değeri 4
i'nin değeri 5
i'nin değeri 6
i'nin değeri 7
i'nin değeri 8
i'nin değeri 9
```

In [4]:
```
# Döngüde i değerlerini ekrana yazdırma

i = 0

while (i < 20):
    print("i'nin değeri",i)
    i += 2 # Koşulun bir süre sonra False olması için gerekli - Unutmayalım
```

```
i'nin değeri 0
i'nin değeri 2
i'nin değeri 4
i'nin değeri 6
i'nin değeri 8
i'nin değeri 10
i'nin değeri 12
i'nin değeri 14
i'nin değeri 16
i'nin değeri 18
```

In [2]:
```
# Ekrana 10 defa "Python Öğreniyorum" yazdıralım.
i = 0

while (i < 10):
    print("Python Öğreniyorum")
    i +=1
```

```
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
Python Öğreniyorum
```

In [3]:
```
# Liste üzerinde indeks ile gezinme
liste = [1,2,3,4,5]

i = 0

while (i < len(liste)):  # len komutu listenin eleman sayısını öğrenmek için
    print("Indeks:",i,"Eleman:",liste[i])
    i +=1
```

```
Indeks: 0 Eleman: 1
Indeks: 1 Eleman: 2
Indeks: 2 Eleman: 3
Indeks: 3 Eleman: 4
Indeks: 4 Eleman: 5
```

-

In [3]:
```
i=0
while i<15:
    print("* " * i)
    i=i+1
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * * *
* * * * * * * * * * * *
* * * * * * * * * * * * *
* * * * * * * * * * * * * *
```

* 'lardan Dik üçgen oluşturma

Sonsuz Döngü Olayları

while döngüsü kullanırken biraz dikkatli olmamızda fayda var. Çünkü, while döngü koşulunun bir süre sonra **False** olması gerekecek ki döngümüz sonlanabilsin. Ancak eğer biz **while** döngülerinde bu durumu unutursak , döngümüz sonsuza kadar çalışacaktır. Biz buna **sonsuz döngü** olayı diyoruz. Örneğimize bakalım

In [ ]:
```
# Bu kodu çalıştırmayalım. Jupyter sıkıntı çıkarabilir :)
i = 0
while (i < 10):
    print(i)
    # i değişkenini artırma işlemi yapmadığımız için i değişkeninin değeri sürekli 0 kalıyor
    # ve döngü koşulu sürekli True kalıyor.
```

Bir sonraki bölümde döngülerde kullanılabilen *range()* fonksiyonu öğreneceğiz.