PYTHON TEMELLERİ

Bu dökümantasyon en az bir programlama dili bilen, Python ’ a hızlı bir giriş yapmak isteyen kişiler için hazırlanmıştır.

Değişkenler

Python’ da değişken tanımlarken açıkça tipi belirtilmez fakat içine verilen default değeri değişkenin tipini belirtir.

```
degisken = 0                #int
degisken = "isim"           #string
degisken = False            #boolean
degisken = 5.5              #float
degisken = [1,2,3]          #list
degisken = (1,2,3)          #tuple
degisken = {1,2,3}          #set
degisken = {'name':"erkan",   #dictionary
            'surname': "görgülü" }
```

Değişkenler ile kullanılabilecek aritmetik operatörler ise şunlardır;

```
+   Addition            x + y   #toplama
-   Subtraction         x - y   #çıkarma
*   Multiplication      x * y   #çarpma
/   Division            x / y   #bölme
%   Modulus             x % y   #mod alma
**  Exponentiation      x ** y  #üs alma örn; 2 ** 5 (2 üzeri 5)
//  Floor division      x // y  #integer bölme (cevap her zaman int değeridir)
```

print()

Ekrana parametre olarak gönderilen değeri ekrana yazar ve bir alt satıra geçer. Dikkat edilmesi gereken bir kural vardır. String ve int tipinde 2 değişkeni kendisi birleştiremez ve TypeError hatası verir.

```
print("isim",5) #doğru yazım şekli
print("isim " + str(5)) #alternatif yazım şekli
print("isim" + 5) #yanlış yazım şekli
```

input()

İnput methodu dışarıdan bir değer almamızı sağlar ve içine parametre olarak string bir değer gönderebiliriz. Gönderdiğimiz değer veri alınmadan önce ekranda gösterilir.

```
name = input("Bir isim giriniz: ") #Değişken string tipinde alınır.
number = int(input("Bir sayı giriniz: ")) #Alternatif olarak bu şekilde de alabiliriz.
```

range()

Range methodu bize belirtilen aralıkta sayı üretir. İçine 3 tane parametre alır. İlki, hangi değerden başlayacağı kendisi de dahil olmak üzere, ikinci parametre ise nerede biteceği fakat kendisi dahil değil. Son parametresi ise değer aralığıdır. Oluşturulan veri kümesi, eğer bir değişken içinde saklanacak ise dizi tiplerinden birine dönüştürülmelidir.(örn; list())

```
listNumbers = list(range(21))       #0 ile 20 arasında
print(listNumbers)
listNumbers = list(range(1,21))     #1 ile 20 arasında
print(listNumbers)
listNumbers = list(range(1,21,2))   #1 ile 20 arasında ikişer ikişer git
print(listNumbers)
```

Üstteki kodun ekran çıktısı ise;

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

if, elif, else

Python’ da if koşulları için parantez kullanılmaz ve neredeyse düz İngilizce yazı şeklinde yazılır. Ama girinti kurallarına dikkat edilmelidir yoksa IndentationError verir. Python, if koşulunu girintiler sayesinde algılar.

```
number = 5
if number > 5:
    print("Sayı 5' ten büyüktür.")
elif number < 5:
    print("Sayı 5' ten küçüktür.")
else:
    print("Sayı 5'e eşittir.")
```

```
fruit = "elma"
if fruit is "elma":
    print("Meyve elmadır.")
elif fruit is not "elma":
    print("Meyve elma değildir")
else:
    print("Bu kod hiç çalışmayacak.")
```

For Döngüsü

Pythondaki for döngüsünde döngü verilen aralık kadar değil içinde dönülen veri kümesinin uzunluğu kadar gider. Basitçe C#’ taki foreach döngüsüne benzer.

```
numbers = list(range(1,21))
for value in numbers:           #veri setindeki her bir değer(value) için
    if value % 2 == 0:          #değer çift sayı ise
        print(value)
```

While Döngüsü

While döngüsü diğer dillerden farklı olarak sadece yazım şeklinde (syntax) değişiklik gösteriyor. Koşul sağlandıkça içinde dönmeye devam eder.

```
num = 5
while num < 5:
    print(num)
    num += 1
```

Veri Kümeleri

Python' da veri kümeleri içinde her türden veri tipini aynı anda barındırabilir. İnt tipinde bir değişken ile string tipinde bir değişken aynı veri kümesi içinde bulunabilir. Fakat bu bazı riskleri beraberinde getirir. Örneğin içinde farklı tiplerde veriler barındıran bir seti dönerken aritmetik işlem yapmak TypeError almamıza sebep olabilir.

```
78   myList = [1,"erkan",2.5,True]
79   total = 0
80   for val in myList:
81       total += val
```

```
Exception has occurred: TypeError
unsupported operand type(s) for +=: 'int' and 'str'
  File "C:\Users\erkan\Desktop\Learn Python\zip,.join().py", line 81, in <module>
    total += val
```

```
82       print(total)
```

List

Listler içinde veri kümesi tutarlar. Diğer dillerdeki listler ile hemen hemen aynı özelliklere sahiptir. En basit haliyle bir list oluşturmak için ihtiyacımız olan tek şey köşeli parantezlerdir([ ]).

```
myList = [1,"erkan",2.5,True]
```

Listlerde indexteki değere erişmek için köşeli parantezler kullanılır.

```
myList = [1,2,3]
print(myList[0])
```

Python'da aynı zamanda negatif indexleme vardır. Bu, değerlere tersten erişim sağlar örneğin listin son elemanına erişmek için;

```
myList = [1,2,3]
print(myList[-1])
```

Listlerde aynı zamanda 2 değerin karşılıklı olarak yerlerini değiştirebileceğimiz hazır bir syntax vardır.

```
myList = ["erkan","görgülü"]
myList[0],myList[1] = myList[1],myList[0]
print(myList)
```

Üstteki kodun çıktısı aşağıdaki gibi olacaktır.

```
['görgülü', 'erkan']
```

List methodları

Listler için Python' da hazır gelen methodlar ise şu şekildedir;

```
myList = []

myList.append(20)               #listenin sonuna 20 değerini ekler

myList.extend([1,2,3,4,5,6])    #listenin sonuna parametre olarak gelen listeyi ekler ve listeyi genişletir.

myList.insert(15,"isim")        #listenin belirtilen indexine değer eklemesi yapar ve
                                #diğer değerleri sona doğru 1 index kaydırır.
                                #Index değeri taşarsa append() gibi sona ekler.

myList.pop()                    #listenin sonundaki elemanı siler.

myList.pop(2)                   #listenin belirtilen indexindeki elemanı siler.

myList.clear()                  #tüm listedeki değerleri siler. değişken boş bir liste halini alır.

myList.remove('isim')           #parametre olarak gönderilen değeri listeden bulup siler, bulamazsa ValueError verir.

myList.index(7)                 #7. indexte bulunan değeri getirir.

myList.index('isim',2,10)       #isim değerinin indexini 2 ve 10. indexler arasında arar
                                #bulursa index değerini döner,bulamazsa ValueError döner.

myList.count(1)                 #1 değerinden listenin içinde kaç tane var onu bulur.

len(myList)                     #listenin içindeki değerlerin adeti için len() methodu kullanılır.

myList.sort()                   #sadece listeyi artarak sıralar ve geriye değer döndürmez.

myList.reverse()                #listenin değerlerini tersine çevirir.
```

Tuple

Tuple’ lar listlerden farklı olarak sadece bir kere tanımlanırlar ve değiştirilemezler. Tuple’ lara ekleme veya çıkarma uygulanamaz. Sadece indexlere erişim vardır. Tuple’ lar sabit oldukları için listlerden daha performanslıdır. Çoğu method geriye değer döndürdüğünde Tuple tipinde döndürür. Tanımlamak için normal parantezler kullanılır.

```
tuples = (1,5,6,3,8,2,0)        #örnek bir tuple tanımlaması
value = tuples[1]               #tuple' daki 1. indexteki değere erişme
tuples.index(5)                 #5 değerinin index ini döndürür
```

Set

Setlerde bir diğer veri kümesi tipidir. İçindeki her bir değer eşsizdir ve aynısından bir tane daha eklenemez. Eklemeye çalışıldığında ise Python değeri görmezden gelir ve eklemez. Sıralı değillerdir. Bu yüzden içindeki değerler hangi sıra ile karşımıza çıkacak bilinmez. Değerler indexlere sahip değildir. Haliyle indexlerle erişim sağlanamaz. Ama içinde döngü ile dönülebilir. Kıvırcık parantez ile tanımlanır.

```
mySet = {1,2,3,4,5,5,5,True,False,0}
print(mySet)
```

Python’ da aynı zamanda 1 değeri True değerine ve 0 değeri de False değerine karşılık gelir. Haliyle üstteki kodun çıktısı aşağıdaki gibi olur. Her zaman ilk yazılan değeri alır sonra gelen aynı değeri görmezden gelir. False değeri 0 ‘ dan önce geldiği için False setin içine girerken 0 giremez.

```
{False, 1, 2, 3, 4, 5}
```

Dictionary

Dictionaryler diğer veri kümelerinden farklı olarak key, value (anahtar, değer) şeklinde çalışırlar. Her bir anahtarın kendisine ait bir değeri vardır. Index yerine anahtarlar üzerinden değere ulaşılır.

```
myDictionary = {'name': 'erkan',
                'surname': 'görgülü',
                'title': 'software developer'
                }
print(myDictionary['name'])
```

Dictionary Methodları

Dictionaryler için Python' da hazır olarak gelen methodlar ise şu şekildedir;

```
myDictionary = {'name': 'erkan',
                'surname': 'görgülü',
                'title': 'software developer'
                }
print(myDictionary['name'])

myDictionary.clear()                #Dictionary' nin içindeki herşeyi siler ve boş bir dictionary' e dönüştürür.

myDictionary.copy()                 #Dictionary' nin bir kopyasını döndürür.

myDictionary.pop('name')            #Dictionary içinden name anahtarını değeri ile birlikte sil.

myDictionary.popitem()              #Dictionary' nin son eklenen anahtar değerini siler.

myDictionary.values()               #Geriye dictionary' nin sadece değerlerini tuple tipinde döndürür.

myDictionary.keys()                 #Geriye dictionary' nin sadece anahtarlarını tuple tipinde döndürür.

myDictionary.items()                #Geriye dictionary' nin tuple tipine dönüştürülmüş halini döndürür.
                                    #Genelde döngüler için kullanılır.

myDictionary.update()               #Dictionary' i girilen parametre ile günceller, bir bakıma yerini alır.

myDictionary.get('name')            #mydictionary['name'] ile aynı işlevi görür

myNewDictionary = dict.fromkeys(['name','surname','age'],"unknown")
                                    #Fromkeys methodu yeni bir dictionary oluşturur ilk girilen liste parametresi
                                    #oluşturulacak anahtarları belirler ve 2. eklenen parametre ise
                                    #anahtarların içinde tutacağı ilk değeri belirler.
```

Comprehensions

List Comprehension

List comprehensionlar Python' da listeleri yaratmak için bir bakıma kısa yol diyebiliriz. Python köşeli parantezler içindeki döngüyü varsa koşula uygun şekilde dönerek bize bir liste döner.

```
myList = [deger**3 for deger in range(1,21) if deger % 2 == 0]
print(myList)
```

Köşeli parantez içinde olan işlemde, 1' den 20' ye kadar olan sayılar içinde, her bir değer için koşul sağlanırsa, yani çift sayi ise, değerin küpünü al ve myList' in içine at. Bu durumda karşımıza aşağıdaki gibi bir sonuç çıkıyor;

```
[8, 64, 216, 512, 1000, 1728, 2744, 4096, 5832, 8000]
```

Dictionary Comprehension

List comprehension' ların yanında aynı zamanda dictionary comprehensionlar da var. Bunlar ise key, value şeklinde döngü içinde dönülüyor ve geriye yine key, value olarak dönülüyor.

```
myDictionary = {deger:deger**2 for deger in range(1,10) if deger % 2 != 0}
print(myDictionary)
```

Yine aynı şekilde 1' den 10' a kadar olan sayılar içinde, eğer değer tek sayı ise, değerin kendisini anahtar olarak, üzeri 2 alınmış halini ise değer olarak myDictionary içine at. Bu işlemden sonra karşımızda şöyle bir sonuç çıkıyor;

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

Functions

Python’ da fonksiyonların dönüş tipi belirtilmez. Parametre alınırken parametrelerin tipi belirtilmez. Bu durum bazı zamanlarda riskler doğursa da çoğunlukla kolaylık sağlar. Fonksiyon tanımlanırken başına def yazılır.

```
def SayHello():
    print("Hello World")

def HelloTo(name = "İsimsiz"):    #Default olarak parametre atanmaz ise İsimsiz kullanılıcak
    print(f"Hello {name}")        #f ile string formatlama python 3 ve üzeri versiyonlarda destekli

print(SayHello())
print(HelloTo("Erkan"))
print(HelloTo())
```

Aynı zamanda fonksiyonun alacağı parametre sayısı belirsiz ise tek bir parametre adı verilip başına * eklendiğinde girilen her değeri bir tuple kümesinde tutar.

```
def AddNumbers(*nums):
    total = 0
    for val in nums:
        total += val
    print(total)

AddNumbers(1,2,3,4,5,6)
```