Python'a Giriş

Python, 1989 yılında oluşturulmaya başlanmış, basit, hızlı, okuması ve yazması kolay bir açık kaynak kodlu programlama dilidir. Şimdiye kadar herhangi bir programlama dilini öğrenen kullanıcılar, kolayca Python da öğrenebilir, ve projelerini Python üzerinden yapabilir.

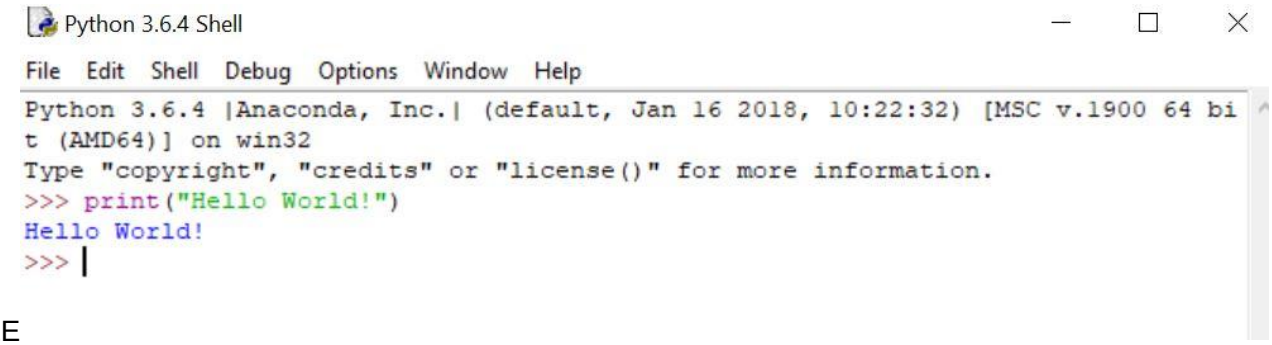
Bu notlarda size … öğretilecektir. Bu bilgiler otomata teorisi projesini yapabilmeniz için yeterli olacaktır.

1. Python Kurulumu

Python'un en çok kullanılan sürümleri 2.7, 3.6.X, 3.7.X ve 3.8.X 'tir. Python 2 ve Python 3 sürümleri arasında ciddi farklar bulunmakla beraber, Python 3'ün kendi içindeki sürümleri (3.6.X, 3.7.X ve 3.8.X olanlar) pek farklı değildir. Biz Python 3'ü kuracağız.

Python'ın istediğiniz sürümünü https://www.python.org/downloads/ adresinden indirilebilirlirsiniz, daha sonra bu indirdiğiniz Python'u next-next .. ile bilgisayarınıza kurabilirsiniz.

python.org resmi adresinden indirdiğiniz Python kendi ide'si ki bunun Python'daki karşılığı IDLE'dir (Integrated Development and Learning Enviroenment), ile birlikte gelir. Bu aşamada eğer herhangi bir Python kodu yazmak isterseniz, windows'a basitçe IDLE yazın, karşınıza çıkar.



IDLE

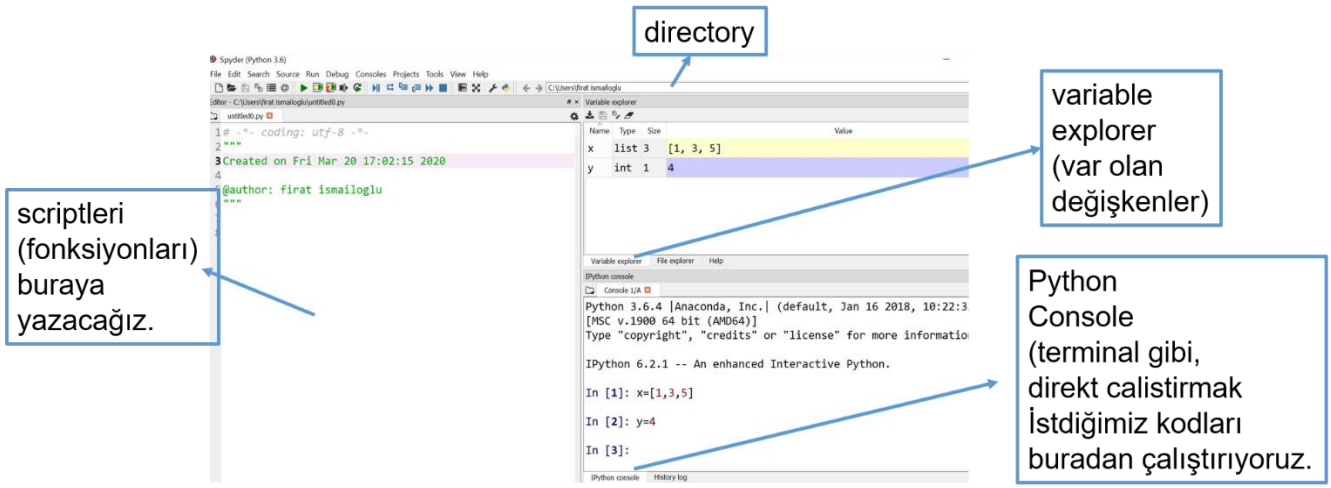
Bir dili IDE ile çalıştırmak çoğu zaman daha efektiftir. Böylece yazdığınız kodu kolayca debug edebilir, varsa yapıtığınız sintaks hatalarını çok daha rahat görebilirsiniz. Python'u indirdikten sonra bir IDE üzerinden çalıştırmak isterseniz bir çok alternatifiniz mevcuttur. Örneğin, Java'dan Eclipse kullanımına alışık olduğunuzdan Eclipse üzerine PyDev kurarak, Python kodlarınızı Eclipse üzerinden de çalıştırabilirsiniz. Yada diğer bir genel environment olan Atom'la birlikte Python çalışabilirsiniz.

Python'a özel environmentlar da mevcuttur. Bunlar içersinde en popüler olanlardan biri PyCharm'dır. Ödevinizde bunu da kullanabilirsiniz, belki bir çoğunuz bilgisayarında halihazırda kuruludur.

Tüm bunların haricinde Python'nu, Anaconda sürümü olarak https://www.anaconda.com/distribution/#download-section adresinden indirebilirsiniz. Bu indirdiğiniz sürümle birlikte, Python ile beraber Jupiter ve Spyder environmentları da gelir. Not: Eğer Anaconda üzerinden Python'ı indirirseniz, tekrar python.org'dan python'u indirmenize gerek kalmaz, Anaconda ile her şey beraber gelir: Python'un kendi, çeşitli IDE'ler..

Kolay arayüzü ve kullanım kolaylığı bakımından bu notta Anaconda ile birlikte gelen Spyder'ı kullanacağım; ama siz ödevinizde herhangi birini kullanmakta özgürsünüz. Spyder'da yazılan bir kod diğer bütün environmentlarda da çalışır.

2. Spyder'ın Tanıtılması



Anaconda indirilip, kurulduktan sonra bilgisayarınızda direkt Spyder'ı aratırsanız, Spyder'ınızı bulursunuz.

Yukarıda Spyder'ın temel pencereleri gösterilmiştir. Buna göre, yazacağımız fonksiyonları soldaki pencreye yazıp, kaydedeceğiz.

Directory, sağda üstte yer almakta. Çağıracağımız, kaydedeceğimiz bütün fonksiyonlar burda yer almak zorunda. Ayrıca varsa okuyacağımız resimler, textler de bu lokasyonda yer almak zorunda.

Variable explorer ile oluşturduğumuz tüm degişkenleri type'ları ve değerleri ile beraber görebiliriz.

Python console ise terminal gibidir, buraya yazdığımız kodları, fonksiyonlari direkt çalıştırabilirz. Yazdığınız uzun bir kodu da burda blok blok çalıştırıp, varsa hatalarınızı bulabilirsiniz.

3. Temel Python Komutları

`input` komutu:

`input` ile kullanicidan girdi aliriz.

```
>> isim=input(" Isminiz nedir? ")
```

```
Isminiz nedir? Firat
```

Artık `isim` değişkeninde "Firat" string'i tutulur.

`print` komutu:

`print` komutu, bildigimiz anlamdaki print komutudur, ekrana metin yazdırmaya yarar.

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

```
Hello World!
```

Kullanicidan aldığımız inputu, print içinde yazdıralım.

```
>> print(" Hello", isim)
```

```
Hello Firat
```

Not: Dikkat edilirse, Python'da değişkenin türünü açıkça ifade etmeye gerek yoktur.

```
>> x = 3
```

Burada x'in type'ı direkt int olarak atanır.

Not: Python'da input komutu ile aldığınız girdiler string tipindedir. Eğer girdiyi bir sayıya dönüştürmek istiyorsak `int()` yada `float()` komutlarını kullanırız.

```
>> sayi=int(input(" Bir sayi giriniz:"))
```

`import` komutu:

`import` komutu ile istdiğimiz kütüphaneyi (eğer bu kütüphane bilgisayara yüklü ise) Python'da çağırabilir daha sonra bu kütüphanenin fonksiyonlarını kullanabiliriz.

```
>> import math      #math kütüphanesini/modülünü cagiriyoruz

>>math.pow(4,2)
```

16

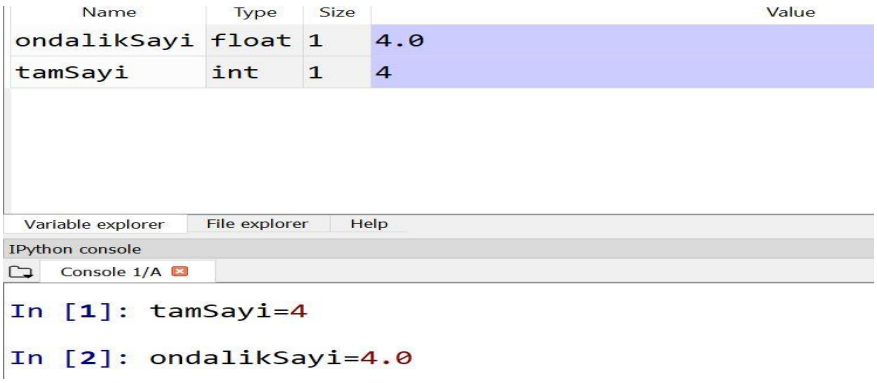
Not: Python'da yorumlar hash (#) ile verilir.

4. Python Veri Tipleri

4.1 Sayısal Veri Tipleri

int : tam sayı

float : ondalık sayı



4.2 Temel Sayı Operatörleri

mutlak değer: `abs(-9)=9`	kalan (mode) `9%5=4`
yuvarlama: `round(8.9)=9`	güç `pow(4,2)=16`

4.3 Mantıksal Operatörler

ve : `& (3>4) & (8<9)= False`	eşitlik `== 9==5=False`
veya: `| (3>4)|(8<9)= True`	değil `! 9!=5= True`

4.4 String Metodları

`lower/upper` metodları:

`.lower()` methodu, eklendiği string'in tüm harflerini küçük harf yapar.

```
>> s="SiVaS"
>> s.lower()
'sivas'
```

Benzer olarak `.upper()` methodu, eklendiği string'in tüm harflerini büyük harf yapar.

```
>> s.upper()
'SIVAS'
```

`split` metodu:

`.split()` methodu eklendiği stringi kelimelerine ayırır.

```
>> ismim="Firat Ismailoglu"
>> ismim.split()
['Firat', 'Ismailoglu']
```

`count` metodu:

`.count("arananan metin")` methodu, eklendigi stringde aranan metin'in kac defa gectigini sayar.

```
>> siir="insanin acisini insan alir"
>> siir.count("insan")
2
```

`startswith/endswith` metodları:

`.startswith("arananan metin")` methodu eklendigi strinde aranan metin ile mi basliyor onu test eder, eger basliyorsa True'ya degilse False'e döner.

```
>> ismim.startswith("Firat")
True
>> ismim.startswith("F")
True
```

`.startswith( )` methodu da benzer şekilde string'in sonunda arama yapmak için kullanılır.

```
>> ismim.endswith("Ismailoglu ")
True
>> ismim.endswith("u ")
True
```

`index` metodu:

`.index("arananan metin")` methodu eklendiği stringde aranan metin kaçıncı karakterden itibaren başlıyorsa o indekse döner.

```
>> siir.index("acisini")
```

Not 1: Python'da indeksler (saymalar) 0'dan baslar. **Not 2**: Python'da string'in çok daha fazla methodu vardır. Bunların tamamına https://www.w3schools.com/python/python_ref_string.asp adresinden ulaşabilirsiniz

4.5 Listeler

list'ler yani listeler çeşitli tipte değişken saklayabileceğiniz yapılardır. Basitçe köşeli parantez [ ] içine yazılırlar.

```
>> listem=[670, 'corona', True]
```

Yukarıda dikkat edilirse üç farkli tipteki değişkeni liste tipinde tutabildik.

4.5.1 Liste Methodları

`append` metodu:

`.append(eklenecek element)` methodu bir listeye yeni bir element eklemeyi (sonuna) sağlar.

```
>> listem.append(223)
[670, 'corona', True, 223]
```

`extend` metodu:

`.extend(eklenecek liste)` methodu ile listeye yeni bir liste ekleyebilriz (yani iki listeyi uc uca ekleyebilriz).

```
>> listem2=['corona', 'el yikama']
>> listem.extend(listem2)
>> print(listem)
[670, 'corona', True, 223,'corona', 'el yikama']
```

`count` metodu:

String'deki count'a benzer olarak, burada da count listede aranan elementin kaç defa geçtiğını sayar.

```
>> listem.count('corona')
2
```

`insert` metodu:

Bir önce gördüğümüz append methodunda eklenen element listenin sonuna ekleniyordu. Insert methoduyla eklenen yeni elementin listenin kaçıncı elemanı olacağına karar verebilriz. Bunun için insert methodun içine önce indeksi yani eklenen elementin yeni listede kaçinci pozisyonda olacagini, sonra ekleyecegimiz elementi yazmamiz gerekir.

```
>> listem.insert(1, 'saglik') # buna göre saglik string'i yeni listedeki indekisi 1 olacak
>> print(listem)
[670, 'saglik','corona', True, 223,'corona', 'el yikama']
```

`index` metodu:

`.index(arananan element)` methodu aranan elementin listenin kacinci elemani oldugunu gosterir.

```
>> listem.index("saglik")
```

Not 1: Bir listede örneğin 3. Sıradaki (indeksi 3 olan) elemani getirmek istiyorsak liste isminden sonra basitce [3] yazariz

```
>> liste[1]
```

```
"saglik"
```

Not 2: Python'da liste veri tipine benzer bir de "tuple" veri tipi vardir. Bu, liste gibi farkli tipten elemanlari bir arada tutmaya yarar; fakat listeden farkli olarak tuple bir kere oluşturulduktan sonra ekleme, çıkarma yapılamaz.

4.5.2 Çok Boyutlu Liste

Python'da liste icinde listeler yazarak cok boyutlu listeler oluşturmak mumkunur.

```
>> liste1=[["Hasan", 22, 176, 70], ["Begum", 21, 160, 52]]
```

```
>> liste1=[1][2] # birinci indis hangi iç liste olacagini, ikinci indis bu listedeki elemani
```

```
160
```

4.5.3 Liste Operasyonları

`len()` listenin uzunluğunu, `min(), max()` listenin sırasıyla en küçük ve en büyük elemanını (tabi listenin tüm elemanları nümerik ise), `[i:j]` listenin i. elemanindan j. elemanına kadar (i dahil, j dahil degil) olan elemanları getirir.

```
>> liste1[0][1:3]
```

```
[22, 176]
```

Pos.index	0	1	2	3	4	5	6
listem	"Merve"	20	170	55	36	0	33
neg.index	-7	-6	-5	-4	-3	-2	-1

Diyelimki `listem=["Merve", 20 , 170, 55, 36, 0, 33]` şeklinde bir listemiz olsun. Bu listenin elementlerinin indeksleri pozitif olarak 0-6 arası, negatif olarak -1 – 7 arasıdır. Dikkat edilirse negatif indeks sondan başa doğru yazılır. Şu halde `listem[0]==listem[-7]; listem[1]== listem[-6];…` ifadelerinin tamamı doğrudur.

Özel olarak eğer listenin son elemaninı cagirmak istersek `listem[-1] yazarız;` ki buda aslinda `listem[len(listem)-1] 'e` denktir.

Not: Python'da string yapılari aslında elemanları bu string'in harfleri olan bir liste gibi düşünebiliriz; fakat bu liste boyutu, içeriği değiştirilemez bir listedir (tuple gibi). Bu şekilde liste tipine ait birçok method, operatör string'ler için de kullanabiliriz.

```
>> ismim="Hasan" # aslinda bu ismim= ["H", "a", "s", "a","n"] listesine denktir
```

```
>> ismim[-1] # son harfi çağırıyoruz
```

```
'n'
```

```
>> len(ismim) # harf sayısı için
```

4.6 Dictionary'ler

Pyhton'daki bir başka veri yapısı dictionary'dir. Dictionary'i iç içe bir çok liste gibi düşünebiliriz; yani listelerin listesi gibi. Fakat burada güzel olan şey içerideki listlere birer isim (key) verebilmek, böylece içerideki hangi listeyi çağıracaksak ismi (key'i) ile çağırabilecek olmamızdır.

Listenin aksine dictionary'ler süslü parantezle gösterilirler. Dictionary'lerin genel yapısı şu şekildedir:

```
{"Key1":[ ,  , , … ], "Key2":[ ,  , , … ], … }
```

Diyelimki şöyle bir yapımız var ve bunu Python'da bir dictionary yapısında tutmak istiyoruz.

Ali	Erkek	26	177	75
Bülent	Erkek	19	190	90
Ceyda	Kadin	22	163	48

```
>> kisiler = {"Ali":["Erkek", 26, 177, 75], "Bülent":["Erkek", 19, 190, 90],
"Ceyda":["Kadin", 22, 163, 48]}
```

Buradan Bülent'e ait bilgilerini çağırmak istersek:

```
>> kisiler["Bülent"]
['Erkek', 19, 190, 90]
```

`.keys()` methodu eklendiği dictionary'deki tüm anahtarları getirir.

```
>> kisiler.keys()
dict_keys(['Ali', 'Bülent', 'Ceyda'])
```

5. Python'da Akışı Kontrol Etmek

5.1 If Statement

Python'da en önemli şeylerden biri bir tab büyüklüğünde oluşturduğunuz boşluktur (ing. indent). Bu boşluklar if bloğunun nerde başlayıp nerde bittiğini belirleyebilirsiz; bu anlamda başka dillerin aksine if'in bloğunu oluşturmak için herhangi bir parantez koymaniza yada end yazmanıza gerek yoktur.

Python'da genel olarak if-else yapısı şu şekilde verilebilir:

```
if <test edilecek mantıksal ifade> :
      if blogu satir 1
      if blogu satir 2
      ...
else:
     else blogu satir 1
      ...
```

Örnek olarak girilen sayı tekse ekrana sayı tek değilse ekrana çift yazdıran program:

```
>> sayi=int(input(" Bir sayi giriniz: "))
>> if sayi %2==1:
      print("Girilen sayi tektir")
   else:
      print("Girilen sayi cifttir")
```

Not: Yeterli uzunlukta boşluk bırakmayı kullanacağınız IDE yardım edecektir.

`elif` , Python'daki else if yapısıdır.

```
>> if sayi <10:
      print("Girilen sayi 10'dan kucuktur")
  elif (sayi>=10) & (sayi<=20):
      print("Girilen sayi 10 ile 20 arasındadır.")
  else:
     print("Girilen sayi 20'den buyuktur.")
```

5.2 For Döngüsü

Python'da for döngüsünün başylayıp bitttiği yer, if'de oldugu gibi boşluklarla sağlanır. Python for loop'ta özel olarak başka bir dilde olmayan `range` kavramı vardır. For'un kaç kere döneceği `range` ile ayarlanır. Genel olarak `range(j, k, l)` demek `j` ile `(k-l)` arası sayıların her bir adımda `l` kadar ilerleyerek gezilmesi demektir.

her bir döngüde `k` kadar artıp, `l'`de sonlanması demektir.

4'ten 10-2' ye kadar olan sayıları ikişer ikişer artarak yazdıralım

```
>> for i in range (4, 10, 2): # 4, 6, 8 sayılarını döndürür.
      print (i)
```

Eğer range'de üçüncü parametre belirlenmesse bu artışın birer bire olacağı anlamına gelir.

```
>> for i in range (4, 10): # 4, 5  ,6, 7, 8, 9 sayılarını döndürür.
      print (i)
```

Eğer range'de ikinci parametre de girilmezsse belirlenmesse, yani tek bir parametre girilirse bu girilen for otomatik olarak 1'den başlar ve birer birer ilerleyerek range'in içindeki değerde sonlanır.

```
>> for i in range (10): # 1,2,3,4,5,6,7,8,9 sayılarını döndürür.
      print (i)
```

Not 1: For loop hiçbir zaman range'in içindeki son değere kadar gitmez, ondan bir önceki değerde biter.

Not 2: For loop'tan erken çıkmak isterseniz break; komutunu kullanabilirsiniz.

Not 3: For loop her zaman range'ile sağlanmaz. Istersek bir list'in, bir tuple'in yada bir dictionary'nin elemanlarini for ile direkt gezebiliriz. Aşağıdaki örnek bu anlamda önemli.

```
>> yeniListe=["sivas", 58 , True]
>> for i in yeniListe: # direkt listenin içinde geziyoruz
      print (i)  # sirasiyla sivas, 58, True yazdirir.
```

Yukarıdaki kişiler adlı disctionary'i hatırlarsak:

```
>> for j in kisiler: # direkt listenin içinde geziyoruz
      print (j)  # sirasiyla Ali, Bülent Ceyda yazdırır.
>> for j in kisiler:
       print(kisiler[i])
['Erkek', 26, 177, 75]
['Erkek', 19, 190, 90]
['Kadin', 22, 163, 48]
```

5.3 While Döngüsü

Python'da while döngüsü for ve if yapılarına benzer; while blogunun neresi oldugu yine boşluklarla sağlanır.

1'den 10'a kadar olan sayıları while ile toplayalım.

```
>> top=0
>> i=0
>> while i<=10:
      top=top+i
      i=i+1    # i+=1 de olurdu
```

Şimdi while'in içine break koyalım.

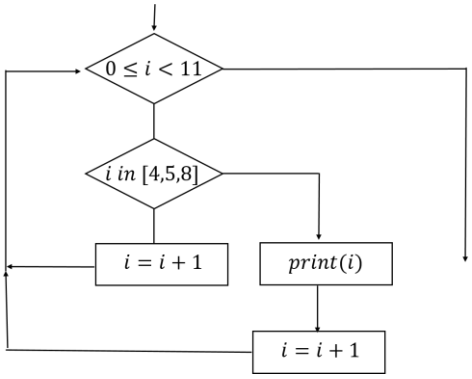
```
>> while i<=10: # 0, 1, 2, 3 yazdırır.
      print(i)
      if i==3:
         break
     i=i+1
```

`continue` komutu:

Python'da istersek for ve while döngülerinde değişkenin bazı değerleri için döngüden "o anlık" çıkabiliriz; yani istediğimiz değişkenler geldiğinde altındaki bloğu çalıştırmayıp, direkt while'a yada for'a dönebiliriz. Bunu, `continue` komutu ile yapariz.

Aşağıdaki örnekte diyelim ki 4 , 5 , 8 değerleri hariç 1 ile 10 arası sayıları yazdıralım.

```
>> for i in range(11):
      if i in [4,5,8]:
         continue # bu çalışırsa alttaki ifade(ler) çlişmaz direkt for'a gidilir.
       print(i)
```



6. Fonksiyonlar

Scriptler, birden cok satirdan oluşan kod bloklaridir. Genelde bir amaç icin yazilirlar. Scriptler

Pyhton'da fonksiyonlar `def` komutu ile tanımlanır (definition'in def'i). Fonksiyonun içeriği yine boşluklarla (indent ile) belirlenir. Fonksiyonun dönderdiği bir değer varsa `return` kullanılır. Bir Python fonksiyonu yazmak iiçin Spyder'da soldaki pencereye, yada herhangi bir text dosyasina aşağıdaki formda fonksiyon yazılır:

```
def <fonksiyon adi>(param1, param2, …) :
       fonksiyon satiri 1
       fonksiyon satiri 2
       ….
```

Yukarıda gösterilenin aksine fonksiyonlar illa parametre(ler) almak zorunda değildir. Eğer oluşturacağınız fonksiyon parametre almayacaksa fonksiyon adından sonra parantez yine açılır, kapanır ama içine bir şey yazılmaz.

```
def ilkFonkum():
      print("Selam Yigido!")
```

Paremetre almayan ve return değeri olmayan fonksiyon

Bu fonksiyonu istersek direkt konsola yazarız. Çağırmak istedğimizde konosola `ilkFonkum()` yazmamız yeterlidir. Yada bu fonksiyonu Spyder'da sol pencereye bir script (kod bloğu) olarak yazıp, working directory'ye kaydederiz. Kaydederken bize script adıini soracak, istediğiniz herhangi bir isim verebilirsiniz; oluşturduğunuz

fonksiyonun adı olmak zorunda değil. Kaydettikten sonra bu script'i Run etmeniz gerekir, bunun için F5 yapin, yada yeşil üçgene basın. Daha sonra fonksiyonunuz kullanılmaya hazır olur.

```
def sayilariTopla(a,b):
      return a+b
```

Paremetreleri ve bir return değeri olan fonksiyon

```
def sayilariToplaCikar(a,b):
      return (a+b),(a-b)
```

Paremetreleri ve iki return değeri olan fonksiyon

Not: Bir scripte birden fazla fonksiyon yazabilirsiniz. Daha sonra bu script'i run ettiginzde tum fonksiyonlar aktif hale gelir.

```
1  # -*- coding: utf-8 -*-
2  """
3
4  """
5  import math #math modulu birçok matematiksel fonk icerir
6
7  def birinciFonksiyon():
8      print("Birinci fonksiyon calisti.")
9
10
11 def ikinciFonksiyon(x):
12     sayininKaresi = pow(x,2)
13     return sayininKaresi
14
15 def ucuncuFonksiyon():
16     sayi=int(input("Bir sayi giriniz"))
17     return math.floor(math.sqrt(sayi))
```

Yukarıda bir script icersinde üç fonksiyon yazılmış. Diyelimki bu scripti yazdınız, ve bunu herhangi bir adla sonu . py ile bitecek şekilde kaydettniz. Örnek olarak ben bunu myScript.py olarak working directory'e kaydettim.

Daha sonra başka bir scriptte buradaki `ikinciFonksiyon` adlı fonksiyonu kullanmak istiyorsunuz. Bunun için yapmanız gereken yeni scriptinizin basina `from myScript import ikinciFonksiyon` yazmaktır.

```
5 from myScript import ikinciFonksiyon
6 ikinciFonksiyon(4)
7
```

Eğer bir scriptteki bütün fonksiyonlari bir baska scriptte kullanmak istiyorsaniz, örnegin myScript'deki `birinciFonksiyon`, `ikinciFonksiyon` ve `ucuncuFonksiyon` 'un hepsini baska bir scriptte kullanmak istersek hepsi anlamindaki * 'i kullaniriz:

```
from  myScript import *
```

Önemli not: Bir scriptten cağıracaginiz butun scriptlerin lokasyonu working directory'nizde olması gerekir!

6.1 main() Fonksiyon

Normalde bir çok dilde hangi sırayla hangi fonksiyonun çalıştırılacağını belirlemek için, akışı tamamen kontrol etmek için, main fonksiyon bulunur. Python'da ise hali hazırda var olan bir main fonksiyonundan bahsedemeyiz; çalıştırılan bir scriptte kodlar direkt yukarıdan aşağıya çalıştırılır. Fakat yinede kodlarin çalıştırılma sırasının düzenlenmesine bazen gerek duyarız. Python'da ise Bunun içinse bildiğimiz anlamda main'in yerine geçebilecek şu ifadeyi script'in herhangi bir yerne yazarız.

```
if __name__=="__main__":
```

Bir script çalıştırıldığında ilk bu if yapısı çalıştırılır; ve bu if her zaman True'ya döner ve sonuç olarak bu if'e ait blok her zaman çalıştırılır. Buradan hareketle bu if bloğunun içinden bir main() fonksiyonu çağıracağız, bu durumda bileceğiz ki script çalıştırıldığında ilk olarak main() fonksiyonu çalışacak. O halde oluşturacağımız main() fonksiyonunun içine ne yazarsak o yazdıklarımız sırayla (yukarıdan aşağıya) çalıştırılacaktır. Şu örneği inceleyelim.

```
1  # -*- coding: utf-8 -*-
2
3
4  def f1():
5      print("Birinci fonksiyon çalıştı.")
6
7  def f2():
8      print("Ikinci fonksiyon calisti")
9
10 def f3():
11     print("Uçuncu fonksiyon calisti.")
12
13 def main():
14     f3()
15     f1()
16     f2()
17
18 if __name__=="__main__":
19     main()
20
```

Bu script çalıştırıldığında aşağıdaki output elde edilir:

```
Uçuncu fonksiyon calisti.

Birinci fonksiyon çalıştı.

Ikinci fonksiyon calisti
```

Dikkat edilirse burada her ne kadar `f1()` birinci fonksiyon olsada, `main()` fonksiyonu onu ikinci sırada çağırdığından `f1()` ikinci sırada çalışmıştır.

6.2 Python'da bir txt dosyası okuma

Pyhton'da bir txt dosyası okumak için open() fonksiyonu kullanılır. Genel olarak bu fonksiyonun kullanım formu:

```
open("okunacakMetininAdı.txt", "r")
```

Burada ikinci parametre olan `r`, read'in r'sidir; metni okumak amaçlı açtğımız anlamına gelir.

Örnek olarak diyelimki working directory'imizde ornekTekst.txt'i adlı bir text dosyamız var ve bunu Python'da açmak istiyoruz.

```
>> met= open("ornekTekst.txt", "r")
```

met adıyla açtığımız bu belgenin içindeki metni çıkarmak için `.read()` methoduna ihtiyacimiz vardir:

```
>> met.read()
```