PYTHON

1) Python, **Guido Van Rossum** adlı Hollandalı bir programcı tarafından 90'lı yılların başında geliştirilmeye başlanmıştır.

2) Adı "The Monty Python" adlı bir İngiliz komedi grubunun, "Monty Python's Flying Circus" adlı gösterisinden esinlenerek konmuştur.

3) Python'ın baş geliştiricisi Guido Van Rossum 2005 ile 2012 yılları arasında Google'da çalışmıştır.

4) Dünya çapında büyük kuruluşlar (Google, YouTube ve Yahoo! gibi) bünyelerinde her zaman Python programcılarına ihtiyaç duyuyor.

5) Python programlama dilini öğrenirseniz, İnternet'te saatlerce ücretsiz PDF birleştirme programı aramak veya profesyonel yazılımlara onlarca dolar para vermek yerine, belgelerinizi birleştirip işinizi görecek programı kendiniz yazabilirsiniz.

6) Python'ı kullanarak masaüstü programlama, oyun programlama, taşınabilir cihaz programlama, web programlama ve ağ programlama gibi pek alanlarda çalışmalar yapabilirsiniz.

7) Python, C ve C++ gibi dillerin aksine derlenmeye ihtiyaç duymaz. Bu yüzden hızlı bir şekilde program geliştirilebilir.

8) Python'ın söz dizimi sade ve basittir. Bu nedenle program yazmak başka dillere kıyasla çok kolaydır.

9) Python kelimesi Türkçe **"paytın"** şeklinde telaffuz edilir.

10) Python pek çok farklı işletim sistemi ve platform üzerinde çalışabilir. GNU/Linux, Windows, Mac OS X, MS-DOS, iOS ve Android gibi belki adını dahi duymadığınız pek çok ortamda Python uygulamaları geliştirebilirsiniz. Ayrıca herhangi bir ortamda yazdığınız bir Python programı, üzerinde hiçbir değişiklik yapılmadan veya ufak değişikliklerle başka ortamlarda da çalıştırılabilir.

11) 1990 yılından bu yana pek çok Python programı üretildiği için piyasada iki çeşit Python sürümü vardır.
Eğer bir Python sürümü 2 ile başlıyorsa (2.7.11 gibi) o sürüm **Python 2.x** serisine aittir. Şayet 3 ile başlıyorsa (3.6.2 gibi) o sürüm Python 3.x serisine aittir.
Python 3.x serisi son yıllarda yaygınlık kazanmaya başlamıştır.

12) Python3, Python2'ye göre hem çok daha güçlüdür, hem de hatalardan arındırılmıştır. Python3'teki büyük değişiklikten ötürü, Python2 ile yazılmış bir program Python3 altında çalışmayacaktır. Aynı durum bunun tersi için de geçerlidir. Yani Python3 ile yazdığınız bir program Python2 altında çalışmaz.

13) Piyasada Python2 ile yazılmış çok sayıda program olduğu için geliştiriciler uzun bir süre daha Python2'yi geliştirmeye devam edecektir.

14) Eğer Python programlama diline yeni başlıyorsanız Python3'ü öğrenmeniz daha doğru olacaktır. Ancak belirli bir proje üzerinde çalışıyorsanız, hangi sürümü öğrenmeniz gerektiği, projede kullanacağınız yardımcı modüllerin durumuna bağlıdır.

15) Şu anda piyasada bulunan bütün Python modülleri/programları henüz Python3'e aktarılmış değildir.

16) Eğer projenizde kullanmayı planladığınız yardımcı modüller hali hazırda Python3'e aktarılmışsa Python3'ü öğrenebilirsiniz. Ancak eğer bu modüllerin henüz Python3 sürümü çıkmamışsa Python2 ile devam etmeniz daha uygun olabilir.

17) Her halükarda Python3'ün bu dilin geleceği olduğunu ve günün birinde Python2'nin tamamen tedavülden kalkacağını da unutmayın.

18) Python'ı kurmak için http://www.python.org/downloads adresine gitmeliyiz.

19) Dowloads linkini tıkladığınızda 'Download Python 3.6.2' ve 'Download 2.7.13' yazan yan yana iki düğme göreceksiniz. İlk düğme Python3, ikinci düğme ise Python2 sürümünü içerir.

20) Python'ı hangi dizine kurduğumuzu bilmek önemlidir. Zira karşılaşacağımız bazı sorunlar Python'ın kurulu olduğu dizine gitmemizi gerektirebilecek veya yazdığımız programlar için orada çeşitli işlemler yapma ihtiyacı duyabileceğiz.

21) Windows kullanıcısı iseniz eğer siz farklı bir yere kaydetmemişseniz Python dosyaları genellikle C:\Users\Kullanıcıadı\AppData\Local\Programs\Python dizini içindedir.

22) Tüm kurulum işlemlerinden sonra Windows kullanıcıları öncellikle komut istemi ile MS-DOS komut satırına gidip 'python' komutunu verirse bu işlem ileride yapacağı çalışmalar açısından daha iyi olacaktır.

23) Asla unutmayın, kullandığınız işletim sisteminin komut satırı ile Python'ın komut satırı birbirinden farklıdır. Yani Windows'ta cmd, Ubuntu'da Ctrl+Alt+T ile ulaştığınız ortam sistem

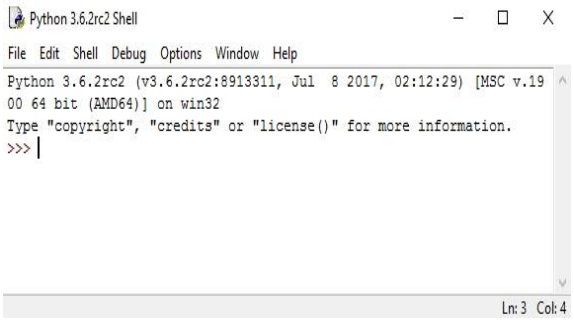
komut satırı iken, bu ortamı açıp python (veya python3 ya da py3) komutu vererek ulaştığınız ortam Python'ın komut satırıdır. Sistem komut satırında sistem komutları (dir, pwr vbs...) Python komut satırında ise Python komutları çalışır.

24) Eğer bilgisayarınıza iki farklı Python sürümü yüklediyseniz bu iki sürümü ayrı ayrı kullanabilirsiniz. Bunun için bir sürümden diğerine geçme komutu vermeniz gerekir. Bu komutlar '**py -2**' ve '**py -3**' komutlarıdır.

25) Komut satırında py-2 komutu verirsek Pyhton'un 2.x versiyonuna geçiş yaparız, py-3 komutu ile de 3.x versiyonuna geçiş yaparız. Ancak daha fazla Python sürümleri varsa bu sefer alt sürüm numaralarını kullanırız. Örneğin py -2.6 komutu verdiğimizde Pyhton 2.6.x sürümüne geçiş yaparız.

26) Python'u normal çalıştırıp editör ortamında veya daha önce değinildiği gibi komut istemiyle ulaştığımız MS-DOS ortamında kullanabiliriz. Bu ortamlara teknik olarak **etkileşimli kabuk adı** verilir.

27) Python Çalışma Ortamları



28) Etkileşimli kabuk dediğimiz komut ortamında **>>> i**şareti Python'da komut yazmaya hazır olunduğu anlamına gelir.

29) Python'a yeni başlayanların en sık yaptığı hatalardan biri >>> işareti ile komut arasında boşluk bırakmalarıdır. Eğer bırakırsanız kod hata verecektir.

30) Hazır Python'a başlamışken programlama kavramlarını hatırlatmakta fayda var. Örneğin **string** kavramı *karakter dizisi* anlamına gelir. Yani bir veya daha fazla karakterden oluşan diziler anlamına gelir. Bunu Python'da örnekleyelim;

>>> "Merhaba Dünya"
'Merhaba Dünya' **//ekran çıktısının tek tırnak içinde olduğuna dikkat edin.**

Burada çift tırnak içinde gösterilen ifade **string**'dir. Çift tırnak önemlidir, çünkü çift tırnak kullanılmazsa program hata verecektir.

31) **type()** komutu ile verilerin tipini sorgulayabiliriz. Yukardaki örneği baz alalım;

```
>>> type("Merhaba Dünya")
<class 'str'>                    //"Merhaba Dünya" adlı veri bir string'tir.
```

32) Karakter dizilerini birleştirmek için aşağıdaki kodu yazmak yeterlidir. Örnek olarak *bilgi* ve *sayar* dizilerini birleştirip ekrana *bilgisayar* yazdırmak istiyorum.

```
>>> "bilgi"+"sayar"
'bilgisayar'                     //+ işaretini kullanarak dizileri birleştirdik.
```

33) Karakter dizilerini birleştirirken boşlukları da kullanmamız gerekebilir. Örneğin Bilgisayar ve Bilimi dizilerini aralarında boşluk bırakarak birleştirelim.

```
>>> "Bilgisayar"+" "+"Bilimi" //ortadaki çift tırnağın içinde bir boşluk bırakıldığına dikkat edin.
'Bilgisayar Bilimi'
```

34) Karakter dizilerini birleştirirken + işaretini kullanmak zorunda değiliz. Yine yukarıdaki örneği baz alalım.

```
>>> "Bilgisayar" " " "Bilimi"
'Bilgisayar Bilimi'
```

//+ yerine hiçbir şey yazmasak ta olur, hatta boşluk bile bırakmak zorunda değiliz.

35) *** (çarpı)** işareti Python'da nasıl bir etki yapar bunu bir örnekle gösterelim.

```
>>> "w" * 3
'www'                            //w stringini üç defa tekrarlayarak ekrana yazdırdı
```

36) **- (eksi) ve / (bölü)** işlemlerini karakter dizileri ile birlikte kullanamayız.

37) Programlama kavramlarını öğrenmeye devam edelim. Örneğin **integer** tam sayı demektir. Yani 96 veya 245 gibi küsuratlı olmayan veriler tamsayıdır.

38) Python'da 8.3 veya 24.18 gibi küsuratlı sayılar **kayan noktalı sayılar** ***(floating point number)*** dediğimiz veri türü sınıfına girer. Ancak burada virgül yerine nokta kullanmamız gerekir.

39) 10+2j gibi bir ifade ise **karmaşık sayılar** (complex) dediğimiz veri türünü oluşturur.

40) Python'u basit bir hesap makinesi gibi kullanabiliriz. Örnekler;

```
>>> 10+5
15

>>> 10*5
50

>>> 10/5
2.0

>>> 10-5
5
```

41) Sayıların çift tırnak içine alınmadığına dikkat edin. Eğer çift tırnak içine alınsaydı veri türü bir sayı değil string olacaktı. Bu durumda matematiksel işlemler yapılamayacaktı. Örnekler;

```
>>> 23+65
88                  // Burada 23 ve 65 bir integer olduğu için matematiksel işlem gerçekleşti.

>>> "23"+"65"
'2365'              // "23" ve "65" birer string'tir. Burada matematiksel işlem yapılamaz.
```

42) **len()** komutu ile parametrelerin uzunluğunu öğrenebiliriz. Bir örnekle açıklayalım.

```
>>> len("Bilgisayar Bilimi")
17

 // "Bilgisayar Bilimi" stringinin uzunluğu 17 karakterdir. Burada boşluk karakterinin de sayıldığına dikkat edelim.
```

43) **len()** komutu ile bir örnek daha yapalım.

```
>>> len("Bilgisayar Bilimi")+ len("Dersi")
22
```

// "Bilgisayar Bilimi" stringi ile "Dersi" stringinin uzunluğu sayısal olarak toplandı.
Yani 17+5= 22 işlemi yapıldı.

44) **len()** komutu çıktı olarak daima sayısal değer verir. Yani len() komutu ile bir **integer** veri türü elde etmiş oluruz.

45) Programlamada sıra geldi **değişken** kavramını öğrenmeye. Değişken matematikte öğrendiğimiz üzere bir değere atama yapmak demektir. Burada amaç, değerler yerine değişkenleri kullanarak daha sade, işlevsel ve zaman kazandırıcı işlemler yapmaktır. Örnek;

```
>>> x=5                  // 5 değerini x değişkenine atadık.
>>> pi=3.14              // 3.14 değerini pi değişkenine atadık.
>>> x+pi                 // Atadığımız değerleri değişkenler yardımıyla topladık.
8.14                     // Sonucu çıktı olarak aldık.
```

46) Değişken atarken bazı kurallara dikkat etmemiz gerekir. Örnekler;

```
5_kilo_elma = "10 tl"      // değişken adları bir sayı ile başlamaz.
+x= 5                      // değişken adları bir aritmetik işleçle başlamaz.
```

47) Değişken adları içinde Türkçe karakterler kullanabilirsiniz. Ancak ileride beklenmedik uyum sorunları çıkması ihtimaline karşı değişken adlarında Türkçe karakter kullanmaktan kaçınmak isteyebilirsiniz.

48) Aşağıdaki kelimeleri değişken adı olarak kullanamayız.

```
   'False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class',
 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for',
 'from',   'global',   'if',   'import',   'in',   'is',   'lambda',   'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield'
```

49) Değişken adı için kullanmamanız gereken yasak kelimeleri Python komut satırında tekrar görmek isterseniz aşağıdaki komutları yazın.

```
>>> import keyword
>>> keyword.kwlist
```

['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

// İşte yasak kelimeleri tekrar gördük. Bunlar özel anlam ifade eden kelimelerdir. Bu yüzden değişken adı olarak kullanılamaz. Kullanılırsa hata mesajı alırız.

50) Yukarıdaki yasak kelimeleri öğrendiniz. Peki, kaç tane yasak kelime var? Tek tek saymak yerine daha önceden verilen len() komutunu kullanabiliriz.

```
>>> len(keyword.kwlist)
33
```

//keyword.kwlist komutuyla oluşturulan listede toplam 33 kelime var.

51) Peki değişken atarken yanlışlıkla yukarıdaki yasak kelimelerden birini kullanırsak ne yapmamız gerekir? Bunun için komut penceresini yani etkileşimli kabuğu kapatıp açabilirsiniz, ya da aşağıdaki komutu kullanabilirsiniz. Örnek olarak *len* komutuna değişken atayalım ve sonra bu yanlıştan dönelim.

```
>>> len=5                  // 5 değerini len değişkenine atadık. (yapmamalıydık)
>>> len("elma")            // elma stringinin uzunluğunu ölçmeye kalktık ama hata aldık
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    type("elma")
TypeError: 'int' object is not callable
```

Şimdi bu hatadan dönüyoruz:

```
>>> del len                                  //işte bu kadar basit
>>> len(“elma”)                              //her şey normale döndü.
4
```

52) Bu arada değişken adlarında asla boşluk olmaz. Örnek;

```
>>>kullanıcı adı = “admin”          // yanlış
>>>kullanıcı_adı = “admin”          // doğru
```

53) Bir sayının kuvvetini bulmak için aşağıdaki kodu yazın.

```
>>> 5**3                  //5'in 3. Kuvvetini hesaplayan kod, iki çarpı işareti yan yana
125

>>> pow(5,3)            //Diğer bir alternatif pow komutunu kullanmak
125
```

54) Bir sayının karekökünü bulmak için aşağıdaki kodu yazın.

```
>>> 144**0.5            //Bir sayının 0.5. kuvveti o sayının kareköküdür
12
```

55) Aşağıdaki kodun anlamı şudur. 11 sayısının 3. kuvveti olan 1331 sayısı 4'e bölündüğünde kalan sayı 3'tür.

```
>>> pow(11,3,4)
3
```

56) Bir değere iki veya daha fazla değişken atayabilirsiniz. Örnek;

```
>>> a=b=4                          //hem a hem de b değişkenine 4 değerini atadık.
>>> a+5
9
>>> b+5
9
```

57) Python'da **değişken takası** gerçekleştirebiliriz. Bir örnekle gösterelim. Bir şirkette Onur isimli kişi ürün müdürü olsun, Murat isimli kişi de insan kaynakları müdürü olsun.

```
>>> onur="ürün_müdürü"
>>> murat="insan_kaynakları_müdürü"
```

Fakat daha sonra bu kişilerin unvanları kendi aralarında değiştiğinde bu işlemi çok basit bir kodla halledebilirsiniz.

```
>>> onur,murat=murat,onur          //işte bu kadar basit
>>> onur                           //onur bundan sonra insan kaynakları müdürü
'insan_kaynakları_müdürü'
>>> murat                          //murat bundan sonra ürün müdürü

'ürün_müdürü'
```

58) _ (alt çizgi işareti), yapılan son işlemin veya girilen son öğenin değerini tutma işlevi görür.

```
>>> 10+5                    // Son işlem olan 15 değerini hafızada tutup bize gösterdi
15

>>> _
15                  // O halde altçizgi ile 5'i topladığımızda sonucun 20 olması kaçınılmaz


>>> _+5
20                          // Bundan sonra _ (yani son öğe) 15 değil 20 olmuştur
```

59) print() fonksiyonu ekrana çıktı vermemizi sağlar. Örnek;

```
>>> print("Merhaba Dünya")
Merhaba Dünya

// İyi ama print fonksiyonu olmadan da çıktı alamıyor muyduk dediğinizi duyar gibiyim.

>>> "Merhaba Dünya"
    'Merhaba Dünya'
```

Bu kodlar da aynı işi görmez mi? Cevap, hayır. Fark ettiyseniz alttaki Merhaba Dünya tek tırnak içine alınmış. Yani aslında bir çıktı değil. İlerde programlarımızı dosyalara kaydedip çalıştırdığımızda, başında print() olmayan ifadelerin çıktıda görünmediğine şahit olacaksınız. Orijinal ekran çıktısı print() fonksiyonu ile gerçekleşir.

60) print() fonksiyonunda çift tırnak kullanmak şart değildir. Yani Merhaba Dünya ifadesini hem çift tırnak hem tek tırnak hem de üç tırnak içerisine alabiliriz. Örnek;

```
>>> print("Merhaba Dünya")                    // çift tırnak
Merhaba Dünya

>>> print('Merhaba Dünya')                   // tek tırnak
Merhaba Dünya

>>> print('''Merhaba Dünya''')               // üç tırnak
Merhaba Dünya
```

Görüldüğü üzere hepsini kullanabiliriz. Ancak Python'da karakter dizisi tanımlarken genellikle tek tırnak ve çift tırnak kullanıldığını unutmayın. O halde üç tırnak neden var sorusuna ise daha sonra yanıt verelim.

61) print() fonksiyonunda neden üç tırnak kullanmamız gerektiğini bir örnek üzerinde gösterelim. Diyelim ki aşağıdaki gibi bir çıktı almak istiyoruz.

```
>>> print("""Python programlama dili Guido Van Rossum
    adlı Hollandalı bir programcı tarafından 90'lı
    yılların başında geliştirilmeye başlanmıştır. Çoğu
    insan, isminin "Python" olmasına bakarak, bu programlama
   dilinin, adını piton yılanından aldığını düşünür.
   Ancak zannedildiğinin aksine bu programlama dilinin
   adı piton yılanından gelmez.""")
```

Böyle bir çıktı alabilmek için tek veya çift tırnak kullanmaya kalkışırsanız epey eziyet çekersiniz. Bu tür bir çıktı vermenin en kolay yolu üç tırnakları kullanmaktır. Çünkü üç tırnaklı yapı öteki tırnak tiplerine göre biraz farklı davranır. Örnek;

```
>>> print('''Game Over
Insert Coin!''')
Game Over
Insert Coin!
```

Yukarıdaki kodların anlamı şu: Ben print komutunu ve üç tırnak işaretini kullanarak Game Over yazdım sonra enter tuşuna bastım, normalde tek tırnak veya çift tırnak kullansaydım enter tuşuna bastıktan sonra program hata verecekti ancak üç tırnak kullandığım için hata vermedi. Bunun anlamı yazmaya devam et demektir. Böylelikle çıktının devamı olan İnsert Coin! ifadesini yazdım ve üç tırnak ve parantez ile fonksiyonumu tamamladım. Tekrar enter tuşuna bastığımda ise üst üste satırlardan oluşan çıktımı elde ettim.

62) Bu arada üç tırnak ''' ya da """ şeklinde olabilir. İkisi de aynı işlevi görür.

63) Peki, neden bazen tek tırnak bazen de çift tırnak kullanmamız gerekiyor? Bunu bir örnekle gösterelim. Diyelim ki aşağıdaki gibi bir çıktı almak istiyoruz.

İstanbul'un 5 günlük hava durumu tahmini

Dikkat ederseniz yukarıdaki ifadede bir kesme işareti var. Yani tek tırnak. O halde kritik soru şu. İçinde tek tırnak işareti olan bir string'i tek tırnak içine almak mantıklı mı? Tabii ki değil çünkü programımız hata verecektir. Bu yüzden diğer bir alternatif olan çift tırnağı kullanmalıyız.

```
>>> print("İstanbul'un 5 günlük hava durumu tahmini")
İstanbul'un 5 günlük hava durumu tahmini
```

64) print() fonksiyonu birden fazla parametre alabilir. Örnek:

```
>>> print ('Fırat' , 'Dicle')                    //virgül kullanarak iki ayrı string'i birleştirdik.
Fırat Dicle
```

65) **sep()** fonksiyonunun ne anlama geldiğini öğrenelim. **sep** ifadesi,
İngilizcede **separator (ayırıcı, ayraç)** kelimesinin kısaltmasıdır. Örnek:

```
>>> print("http://", "www.", "google.", "com")
http:// www. google. com
```

// Yukarıdaki örnekte virgül işareti ile stringleri birleştirirken birer boşluk bırakıldığını fark etmişsinizdir. Ancak boşluk bırakılmasını her zaman istemeyiz. Yukarıdaki linkte boşluk olmaması gerekir. Yani bizim hedefimiz http://www.google.com yazdırmak. Bildiniz gibi linklerin boşluklu hali hiçbir işe yaramaz. İşte burada sep() fonksiyonu devreye girer. Aslında yukarıdaki kod şu şekildedir:

```
>>> print("http://", "www.", "google.", "com", sep=" ")
```

// Ancak buradaki **sep=" "** ifadesi görünmezdir yani onu normalde kullanmayız o ancak arka planda çalışır ve default olarak virgüllerden sonra boşluk bırakır, ta ki amacımız değişene kadar. Çift tırnak işaretinin arasına boşluk değil de başka bir karakter koyarsak bu sefer virgüllerden sonra boşluk değil bu koyduğumuz karakter olacaktır. O halde çift tırnak arasına bir şey koymazsak virgüllerden sonra hiçbir boşluk olmayacaktır. Doğru kodumuzu yazalım.

```
>>> print("http://", "www.", "google.", "com", sep="")          //Bir boşluk nelere kadir

http://www.google.com
```

66) **sep=" "** ile **sep=None** aynı anlama gelir.

67) bir mumdur iki mumdur üç mumdur dört mumdur… türküsünü sep ve print fonksiyonu kullanarak ekrana yazdıralım.

```
>>>print("bir", "iki", "üç", "dört", "on dört", sep=" mumdur ")
bir mumdur iki mumdur üç mumdur dört mumdur on dört
```

//virgüller arasına boşluk, mumdur ve tekrar boşluk koyduk. Yani sep=" mumdur "

68) **print()** fonksiyonunun **end** adlı özel bir parametresi daha bulunur. Tıpkı **sep** parametresi gibi, end parametresi de print() fonksiyonunda görünmese bile her zaman oradadır. sep parametresi print() fonksiyonuna verilen parametreler birleştirilirken araya hangi karakterin gireceğini belirliyordu. end parametresi ise bu parametrelerin sonuna neyin geleceğini belirler. Bunu bazı örneklerle açıklayalım.

```
>>> print("Bilgisayar\nBilimi")
Bilgisayar
Bilimi
```

\n parametresini far etmişsinizdir. Bu parametreye **newline** adı verilir. Görevi de yukarıda görüleceği gibi satırbaşı yaparak stringleri ayırmaktır. Örnek:

```
>>> print("Bugün günlerden Salı")
Bugün günlerden Salı
```

//Burada herhangi bir end parametresi göremiyoruz. Ancak Python yukarıdaki kodu aslında şöyle algılar:

```
>>> print("Bugün günlerden Salı", end="\n")
```

//Kısacası bu kodu yazdığımızda ve enter tuşuna bastığımızda print() fonksiyonu iki farklı işlem gerçekleştirir:

1-Öncelikle karakter dizisini ekrana yazdırır.
2-Ardından bir alt satıra geçip bize >>> işaretini gösterir.

//Bunun ne demek olduğunu anlamak için end parametresinin değerini değiştirmemiz yeterli olacaktır:

```
>>> print("Bugün günlerden Salı", end=".")
Bugün günlerden Salı.
```

// end fonksiyonu ile stringin sonuna bir nokta koymuş olduk. Aslında mesele bu kadar basit.

69) **sys.stdout** nedir? sys.stdout **'standart çıktı konumu'** anlamına gelir. Standart çıktı konumu; bir programın, ürettiği çıktıları verdiği yerdir. Python öntanımlı olarak, ürettiği çıktıları ekrana verir. Eğer o anda etkileşimli kabukta çalışıyorsanız, Python ürettiği çıktıları etkileşimli kabuk üzerinde gösterir. Dolayısıyla Python'ın standart çıktı konumu etkileşimli kabuk veya komut satırıdır. Yani print() fonksiyonu yardımıyla bastığınız çıktılar etkileşimli kabukta ya da komut satırında görünecektir. Ama eğer istersek print() fonksiyonunun, çıktılarını ekrana değil, bir dosyaya yazdırmasını da sağlayabiliriz. Bunu daha sonraki örneklerle gösterelim.

70) **print()** fonksiyonu ile elde edilen çıktıları ekrana değil de bir dosyaya nasıl yazdırırız?

```
>>> dosya = open("deneme.txt", "w")
>>> print("Bilgisayar Bilimi, Python",file=dosya)
>>> dosya.close()
```

Gelin yukarıdaki kodları açıklayalım. Öncelikle **deneme.txt** adlı bir dosya oluşturduk ve bunu dosya adlı bir değişkene atadık. Burada kullandığımız **open()** fonksiyonuna çok takılmayın. Biz şimdilik bu şekilde dosya oluşturulduğunu bilelim yeter. Gördüğünüz gibi **open()** fonksiyonu da birtakım parametreler alabiliyor. Bu fonksiyonun ilk parametresi **"deneme.txt"** adlı bir karakter dizisi. İşte bu karakter dizisi bizim oluşturmak istediğimiz dosyanın adını gösteriyor. İkinci parametre ise "w" adlı başka bir karakter dizisi. Bu da **deneme.txt** dosyasının yazma kipinde (modunda) açılacağını gösteriyor. Oluşturduğumuz bu **deneme.txt** adlı dosya, o anda bulunduğunuz dizin içinde oluşacaktır. Bu dizinin hangisi olduğunu öğrenmek için şu komutları yazın:

```
>>> import os
>>> os.getcwd()
'C:\\Users\\Kullaciadi\\AppData\\Local\\Programs\\Python\\Python36'
```

Dizine baktığınızda **deneme.txt** adlı bir dosya oluştuğunu göreceksiniz. Dosyayı açtığınızda Bilgisayar Bilimi, Python çıktısının yazılı olduğuna şahit olacaksınız. Eğer dizin görünmüyorsa gizli olabilir bu durumda gizli dosyaları göster deyin.

Yukarıdaki kodu açıklamaya devam edelim. Gördüğünüz üzere ikinci satırda **print()** fonksiyonuna rağmen ekranda çıktı görünmedi. Çünkü **print()** fonksiyonunu ekrana değil dosyaya çıktı verecek şekilde ayarladık. Bu işlemi **file** adlı bir parametreye, dosya değişkenini atayarak yaptık. Son komut yardımıyla da, yaptığımız değişikliklerin dosyada görünebilmesi için ilk başta açtığımız dosyayı kapattık.

71) Tıpkı **sep** ve **end** parametreleri gibi, file parametresi de, siz görmeseniz bile her zaman print() fonksiyonunun içinde yer alır. Yani diyelim ki şöyle bir komut verdik:

```
>>>print("Tahir olmak da ayıp değil", "Zühre olmak da")
```

Python aslında bu komutu şöyle algılar:

```
>>> print("Tahir olmak da ayıp değil", "Zühre olmak da", sep=" ", end="\n",
file=sys.stdout)
```

Yani parametrelerin arasına birer boşluk koyar **(sep=" "),** sonuna satır başı karakteri ekler **(end="\n")** ve bu çıktıyı standart çıktı konumuna gönderir **(file=sys.stdout).**

72) **flush()** fonksiyonunu aşağıdaki örnekle açıklayalım.

```
>>> f=open("kisisel_bilgiler.txt","w")
>>> print("Murat Yılmaz", file=f)
>>> print("T.C. no: 56897136119", file=f)
>>> print("Doğum yeri: Trabzon", file=f)
>>> f.close()
```

Yukarıdaki kodları daha önce öğrenmiştik. **kisisel_bilgiler** adlı bir txt dosyası oluşturduk. Ardından bu dosyaya sırasıyla yukarıda yazılı olan çıktıları gönderdik. Son olarak bu dosyayı kapatarak işlemi sonlandırdık. Ancak bu son dosyayı kapatma işlemi olmadan yukarıdaki çıktıları almamız mümkün. İşte flush() fonksiyonu burada devreye giriyor. O halde flush() fonksiyonunu kullanarak yeniden kodları yazalım.

```
>>> f=open("kisisel_bilgiler.txt","w")
>>> print("Murat Yılmaz", file=f, flush=True)
>>> print("T.C. no: 56897136119", file=f, flush=True)
>>> print("Doğum yeri: Trabzon", file=f, flush=True)
```

Burada yaptığımız şey sadece print() fonksiyonunun içine **flush=True** değerini koymak. flush() fonksiyonunun sadece iki değeri vardır. **True or False**. flush() değeri default olarak False ayarlandığı için yapmak istediğimiz işlem için True değerini kendimiz vermek zorunda kaldık. Artık kodun sonuna **f.close()** koymak zorunda değiliz.

73) ***(yıldız)** parametresini anlatmak için bir soru soralım. Aşağıdaki gibi bir çıktı almak istersek nasıl bir kod yazmamız gerekir?

```
L.i.n.u.x
```

Aslında cevabı daha önce öğrenmiştik.

```
>>>print("L", "i", "n", "u", "x", sep=".")
L.i.n.u.x
```

Ancak bunun daha basit bir yolu var.

```
>>>print(*"Linux", sep=".")
 L.i.n.u.x
```

Bu örneklerden de gördüğünüz gibi * işaretini kullanarak Linux stringini parçalara böldük ve bu parçaların arasına sep() fonksiyonunu kullanarak nokta koyduk. Bu kadar basit.

74) **sys.stdout** modülünü (standart çıktı konumu) kalıcı olarak değiştirmek için aşağıdaki kodu yazalım.

```
>>> import sys
>>> f =open("ödevler.txt", "w")
>>> sys.stdout=f
>>> print("İstanbul'un Fethi Slaytı", flush=True)
```

Yukarıdaki kodu açıklayalım. İlk önce **import sys** komutu vererek sys modülünü içe aktardık. import sys önemli çünkü bu komutu vermezseniz bundan sonra sys ile ilgili yazacağınız hiçbir kod çalışmayacaktır. Bunu modüller konusu işlenirken daha iyi anlayacağız. Daha sonra **ödevler.txt** adlı bir dosyayı **"w"** komutunu da kullanarak yazma kipinde oluşturduk ve bunu f değişkenine atadık. Asıl can alıcı kodu ise **sys.stdout=f** ile oluşturduk çünkü kalıcı dosyayı f değişkenine atayarak bundan sonra yazacağımız her şeyi ama her şeyi **ödevler.txt** dosyasına aktarma emri verdik. Son satırda ise basit bir **print()** emri vererek ödevler.txt dosyasına "İstanbul'un Fethi Slaytı" yazısını yazdırmış olduk. Haliyle dosyayı kapatmaya ihtiyacı duymadık çünkü hemen akabinde **flush=True** değerini oluşturduk.

75) Peki standart çıktı konumunu, etkileşimli kabuktan çıkmadan veya programı kapatmadan eski haline döndürmenin bir yolu var mı? Elbette var. Yukarıdaki koda aşağıdaki kodu ekleyelim.

```
f, sys.stdout = sys.stdout, f
```

İşte bu kadar basit. Yaptığımız şey değişken takasından başka bir şey değil.

76) Son olarak aşağıdaki kodları açıklayarak **sys.stdout** konusunu daha iyi anlayalım.

```
>>> sys.stdout.name
>>> sys.stdout.mode
>>> sys.stdout.encoding
```

Burada **sys.stdout.name** komutu standart çıktı konumunun o anki adını verecektir. sys.stdout.mode komutu ise standart çıktı konumunun hangi kipe sahip olduğunu gösterir. Standart çıktı konumu genellikle yazma kipinde (w) bulunur. **sys.stdout.encoding** kodu ise standart çıktı konumunun sahip olduğu kodlama biçimini gösterir. Kodlama biçimi, standart çıktı konumuna yazdıracağınız karakterlerin hangi kodlama biçimi ile kodlanacağını belirler. Kodlama biçimi Windows'ta genellikle **'cp1254',** GNU/Linux'ta ise **'utf-8'**dir. Eğer bu kodlama biçimi yanlış olursa, mesela dosyaya yazdıracağınız karakterler içindeki Türkçe harfler düzgün görüntülenemez.

77) **Kaçış dizileri (escape sequence)** nedir? Python'da özel anlam taşıyan işaret veya karakterleri, sahip oldukları özel anlam dışında başka bir amaçla kullanmamızı sağlayan araçlardır. Örnek;

```
>>> print("Python programlama dilinin adı \"piton\" yılanından gelmez")
Python programlama dilinin adı "piton" yılanından gelmez
```

Yukarıdaki karakter dizisini hem çift tırnak içine alıp hem karakter dizisi içinde çift tırnak kullandığımız halde herhangi bir hata almadığımızı görüyorsunuz. Yukarıdaki kodlarda hata almamızı önleyen şey **\ (ters taksim)** işaretidir. **\ (ters taksim)** kaçış dizilerinden sadece bir tanesidir. Başka bir örnek:

```
>>> print("Python 1990 yılında Guido Van Rossum \
tarafından geliştirilmeye başlanmış, oldukça \
güçlü ve yetenekli bir programlama dilidir.")
Python 1990 yılında Guido Van Rossum tarafından geliştirilmeye başlanmış, oldukça güçlü
ve yetenekli bir programlama dilidir.
```

Normal şartlar altında, bir karakter dizisini tanımlamaya tek veya çift tırnakla başlamışsak, karakter dizisinin kapanış tırnağını koymadan Enter tuşuna bastığımızda Python bize bir hata mesajı gösterir. İşte **\ kaçış dizisi** bizim burada olası bir hatadan kaçmamızı sağlar. Eğer Enter tuşuna basmadan önce bu işareti kullanırsak Python tıpkı en üst örnekteki çift tırnak işaretinde şahit olduğumuz gibi hata vermeden bir alt satıra geçecektir.

78) Bazen kaçış dizilerini fark etmemiz gerekir çünkü kodumuz hata verebilir. Örnek:

```
>>> open("C:\nisan\masraflar.txt")
Traceback (most recent call last):
File "<pyshell#4>", line 1, in <module>
open("C:\nisan\masraflar.txt")
OSError: [Errno 22] Invalid argument: 'C:\nisan\\masraflar.txt'
```

Yukarıdaki kodumuz hata verdi. Bir dosya açmaya çalıştık ama dosyanın içindeki \n ifadesini gözden kaçırdık. Bu aslında bir kaçış dizisidir ve daha önceki konularda gördüğümüz üzere satırbaşı yapmaya yarar. Eğer sorunun gözden kaçan bir kaçış dizisinden kaynaklandığını fark edemezseniz, bu sorunu çözebilmek için saatlerinizi ve hatta günlerinizi harcamak zorunda kalabilirsiniz. Ancak ve ancak yukarıdaki karakter dizisi içinde sinsice gizlenen bir \n kaçış dizisi olduğu gözünüze çarparsa bu sorunu çözme yolunda bir adım atabilirsiniz. Bu problemi en başta kullandığımız kaçış dizisi olan **** işareti ile çözebiliriz.

```
>>>open("C:\\nisan\\masraflar")    // bir \ işareti ile \n parametresini yok saydık.
>>>open("C:/nisan/masraflar")     // ya da dizin adlarını ters taksim yerine düz taksim ile ayırmayı tercih edebiliriz.
```

79) Sekme (\t) kaçış dizisi:

```
>>> print("abc\tdef")
abc     def
```

Ne olduğunu fark ettiniz mi? **\t** parametresi “abc” ifadesinden sonra sanki Tab (sekme) tuşuna basılmış gibi “def” ifadesini sağa doğru itiyor.

```
>>> print(*"123456789", sep="\t")
1       2       3       4       5       6       7       8       9
```

Peki, şimdi ne yaptık? String karakterlerinin arasına belli aralıklarla boşluk bıraktık.

80) Zil sesi (\a) kaçış dizisi:

```
>>> print("\a")                                    // bir bip sesi elde ettik.
!bip!

 >>> print("\a"*5)                                // bip sesini 5 katına çıkardık.
!bip!!bip!!bip!!bip!!bip!
```

Bu kaçış dizisinin GNU/Linux üzerinde çalışma garantisi yoktur. Hatta Windows sistemlerinde dahi çalışmayabilir. Hatta burada çıktı olarak **!bip!** ifadesinin yerine sembolü gelebilir.

81) Aynı satırbaşı (\r) kaçış dizisi:

```
>>> print("Merhaba\rDünya")
Dünyaba
```

Burada, "Merhaba" karakter dizisi ekrana yazdırıldıktan sonra **\r** kaçış dizisiyle satır başına dönülüyor ve bu kaçış dizisinden sonra gelen "Dünya" karakter dizisi "Merhaba" karakter dizisinin üzerine yazıyor. Tabii "Dünya" karakter dizisi içinde 5 karakter, "Merhaba" karakter dizisi içinde ise 7 karakter olduğu için, "Merhaba" karakter dizisinin son iki karakteri ("ba") dışarda kalıyor. Böylece ortaya "Dünyaba" gibi bir şey çıkıyor. Bu kaçış dizisinin her işletim sisteminde çalışma garantisi yok. Ancak yine de bilmek te fayda var.

82) Düşey sekme(\v) kaçış dizisi:

```
>>> print("düşey\vsekme")     // alt satıra geçip tab tuşuna basılmış etkisi yarattı.
düşey
    sekme
```

Her işletim sisteminde çalışmayabilir. Dolayısıyla önerilmez.

83) İmleç Kaydırma (\b) kaçış dizisi:

```
>>> print("yahoo.com\b.uk")
yahoo.co.uk
```

\b kaçış dizisinin etkisiyle imleç bir karakter sola kaydığı için, 'com' kelimesinin son harfi silindi ve bunun yerine **\b** kaçış dizisinden sonra gelen .uk karakterleri yerleştirildi. Bu kaçış dizisinin Python'da çok nadir kullanıldığını ve çalışmama ihtimalini söyleyelim.

84) Küçük Unicode (\u) kaçış dizisi:

UNICODE, karakterlerin, harflerin, sayıların ve bilgisayar ekranında gördüğümüz bütün işaretlerin her biri için tek ve benzersiz bir numaranın tanımlandığı bir sistemdir. Bu sistemde, **'kod konumu' (code point)** adı verilen bu numaralar özel bir şekilde gösterilir. Örneğin 'İ' harfi UNICODE sisteminde şu şekilde temsil edilir:

u+0130

Python programlama dilinde yukarıdaki kod konumu ise şöyledir:

\u0130

O halde kodumuzu yazalım ve çıktıyı görelim.

```
>>> '\u0130'
'İ'
```

85) Büyük Unicode (\U) kaçış dizisi

Bu kaçış dizisi \u adlı kaçış dizisiyle hemen hemen aynı anlama gelir. Ancak **U** ile gösterilen kod konumları **u** ile gösterilenlere göre biraz daha uzundur. Küçük u kaçış dizisini kullanarak **'İ'** harfinin UNICODE kod konumunu şöyle temsil ediyorduk:

```
>>> '\u0130'
'İ'
```

Eğer aynı kod konumunu büyük U ile göstermek istersek şöyle bir şey yazmamız gerekir:

```
>>> '\U00000130'
'İ'
```

Gördüğünüz gibi, burada **\U** kaçış dizisinden sonra gelen kısım toplam 8 haneli bir sayıdan oluşuyor. **\u** kaçış dizisinde ise bu kısmı toplam 4 haneli bir sayı olarak yazmıştık. İşte \u kaçış dizisi ile **\U** kaçış dizisi arasındaki fark budur.

86) Uzun Ad (\N) kaçış dizisi: UNICODE sisteminde her karakterin benzersiz bir kod konumu olduğu gibi bir de uzun adı vardır. Örneğin **'a'** harfinin UNICODE sistemindeki uzun adı şudur: LATIN SMALL LETTER A

Bu uzun adları öğrenmek için **unicodedata** adlı bir modülden yararlanabilirsiniz:

```
>>>import unicodedata     // Bir modül kullanmak için import komutu hayatidir unutmayın
>>>unicodedata.name('a')
'LATIN CAPITAL LETTER A WITH CEDILLA'
```

İşte \N kaçış dizisi bu uzun isimleri kullanarak asıl karakterleri elde eder.

```
>>> print("\N{LATIN SMALL LETTER A}")
A
```

87) Onaltılı Karakter (\x) kaçış dizisi: Onaltılı (hexadecimal) sayma sistemindeki bir sayının karakter karşılığını gösterir.

```
>>> "\x41"              // Onaltılı sayma sistemindeki 41 sayısı 'A' harfine karşılık gelir.
'A'
```

88) Etkisizleştirme (r) kaçış dizisi: Bu kaçış dizisini, daha önce öğrendiğimiz \r adlı kaçış dizisi ile karıştırmayın. Şöyle bir çıktı almak istediğimizi düşünün;

Kurulum dizini: C:\aylar\nisan\toplam masraf

```
>>> print("Kurulum dizini: C:\aylar\nisan\toplam masraf")
        Kurulum dizini: C:ylar
isan      oplam masraf
```

Olmadı dimi? Python karakter dizisi içinde geçen '**\a**ylar', '**\n**isan', ve '**\t**oplam masraf' ifadelerinin ilk karakterlerini yanlış anladı! Bu gibi ifadeler Python'ın gözünde birer kaçış dizisi. Dolayısıyla Python \a karakterlerini görünce bir !bip! sesi çıkarıyor, \n karakterlerini görünce satır başına geçiyor ve \t karakterlerini görünce de Tab tuşuna basılmış gibi bir tepki veriyor. Daha önce bu durumu şöyle bir kod yazarak engellemiştik:

```
>>> print("Kurulum dizini: C:\\aylar\\nisan\\toplam masraf")
Kurulum dizini: C:\aylar\nisan\toplam masraf
```

Ama bu sorunun üstesinden gelmenin çok daha basit ve pratik bir yolu vardır:

```
>>> print(r"Kurulum dizini: C:\aylar\nisan\toplam masraf")
Kurulum dizini: C:\aylar\nisan\toplam masraf      // baştaki r harfi her şeyi hallediyor.
```

89) Amacımız aşağıdaki gibi bir çıktı elde etmek olsun.

**Kaçış dizisi: **

O halde kodumuzu yazalım.

```
>>>print("Kaçış dizisi: \")
SyntaxError: EOL while scanning string literal
```

Durup dururken bu kod neden çalışmadı acaba? Biliyorsunuz ki çift tırnak açtığımız zaman tekrar kapatmamız gerekiyor. Ancak ilginç bir şekilde burada çift tırnağımızı kapattığımız görülüyor. Yani sorun yok gibi. Ama öyle değil, lütfen sondaki kapanış tırnağının soluna bakın. Ne görüyorsunuz? **** işareti dimi? Peki bu işaretin anlamı neydi? Bu işaret önündeki parametreleri geçersiz kılıyordu. Yani bir nevi yok edici. Bu işaret yüzünden sanki kapanış tırnağını kapatmamışız gibi bir etki yarattık. Hatta öyle ki bu etkisizleşmeyi, karakter dizisinin başına koyduğunuz **'r'** kaçış dizisi de engelleyemez. O halde çözüm nedir? Birinci çözüm **** işaretinin sağına bir adet boşluk karakteri yerleştirmektir. Çünkü **** işareti sağına baktığı zaman sadece bir boşluk görecek ve etkisizleştirecek bir şey bulmayacak. Örnek:

```
>>> print("Kaçış dizisi: \ ")          \ işaretinin hemen sağında bir boşluk bıraktık.
Kaçış dizisi: \
```

Diğer bir çözüm kaçış dizisini çiftlemek. Yan yana ters taksim yapacağız. Böylece birbirini götürecekler ve bir etki olmayacak.

```
>>> print("Kaçış dizisi: \\")
Kaçış dizisi: \
```

Ya da karakter dizisi birleştirme yöntemlerinden herhangi birini kullanabilirsiniz:

```
>>> print("Kaçış dizisi: " + "\\")
>>> print("Kaçış dizisi:", "\\")
>>> print("Kaçış dizisi: " "\\")
```

90) Sayfa Başı (\f) kaçış dizisi: Pek kullanılmayan bir kaçış dizisidir. Görevi, özellikle eski yazıcılarda, bir sayfanın sona erip yeni bir sayfanın başladığını göstermektir. Eski model yazıcılar, bu karakteri gördükleri noktada mevcut sayfayı sona erdirip yeni bir sayfaya geçer.

```
>>> f = open("deneme.txt", "w")
>>> print("deneme\fdeneme", file=f)
>>> f.close()
```

Şimdi bu kodlarla oluşturduğunuz deneme.txt adlı dosyayı Microsoft Word gibi bir programla açın. 'deneme' satırlarının iki farklı sayfaya yazdırıldığını göreceksiniz. Bu arada, eğer Microsoft Word dosyayı açarken bir hata mesajı gösterirse, o hata mesajına birkaç kez 'tamam' diyerek hata penceresini kapatın. Dosya normal bir şekilde açılacaktır.

91) **Programları Kaydetme ve Çalıştırma:** Şimdiye kadar bütün işlerimizi Python'ın etkileşimli kabuğu üzerinden hallettik. Ancak etkileşimli kabuk son derece kullanışlı bir ortam da olsa, bizim asıl çalışma alanımız değildir. Kodlar çoğalıp büyümeye başlayınca bu ortam yetersiz gelmeye başlayacaktır.

92) Etkileşimli kabuğu genellikle ufak tefek Python kodlarını test etmek için kullanırız.

93) Asıl programlarımızı etkileşimli kabuğa değil, program dosyasına yazarız.

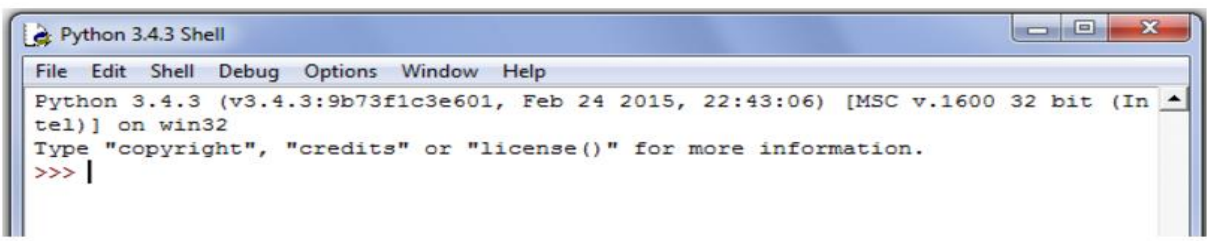
94) Bu arada yazdığınız kodları bir yere kaydedip saklamak isteyeceksiniz. İşte burada metin düzenleyiciler devreye girecek.

95) Python kodlarını yazmak için istediğiniz herhangi bir metin düzenleyiciyi kullanabilirsiniz. Hatta **Notepad** bile olur. Ancak Python kodlarını ayırt edip renklendirebilen bir metin düzenleyici ile yola çıkmak her bakımdan hayatınızı kolaylaştıracaktır.

96) Python kodlarınızı yazmak için Microsoft Word veya OpenOffice.Org OOWriter gibi, belgeleri ikili (binary) düzende kaydeden programlar uygun değildir. Kullanacağınız metin düzenleyici, belgelerinizi **düz metin (plain text)** biçiminde kaydedebilmeli.

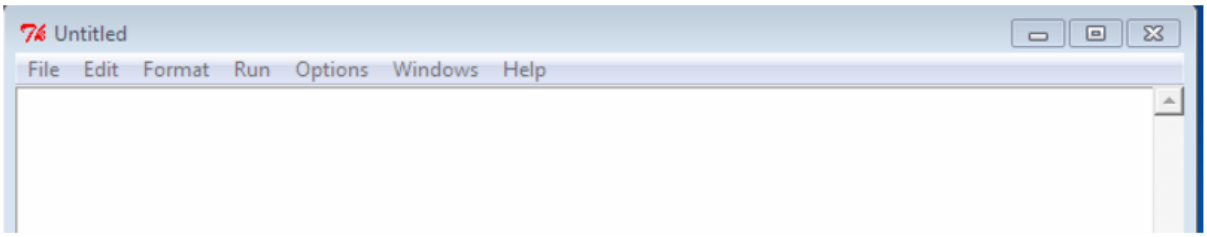
97) Eğer kullandığınız sistem GNU/Linux'ta Unity veya GNOME masaüstü ortamı ise başlangıç düzeyi için Gedit adlı metin düzenleyici yeterli olacaktır. Eğer kullandığınız sistem GNU/Linux'ta KDE masaüstü ortamı ise Kwrite veya Kate adlı metin düzenleyicilerden herhangi birini kullanabilirsiniz. Şu aşamada kullanım kolaylığı ve sadeliği nedeniyle **Kwrite** önerilebilir.

98) Python programlama dilini öğrenmeye yeni başlayan Windows kullanıcıları için en uygun metin düzenleyici **IDLE'dır.** *Başlat > Tüm Programlar > Python*3.5 *> IDLE (Python GUI)* yolunu takip ederek IDLE'a ulaşabilirsiniz. **IDLE aslında Python'ın etkileşimli kabuğudur. Dolayısıyla asıl kodları buraya yazmayacağız.** Python programlama diline yeni başlayanların en sık yaptığı hatalardan biri de, kaydetmek istedikleri kodları bu ekrana yazmaya çalışmalarıdır.



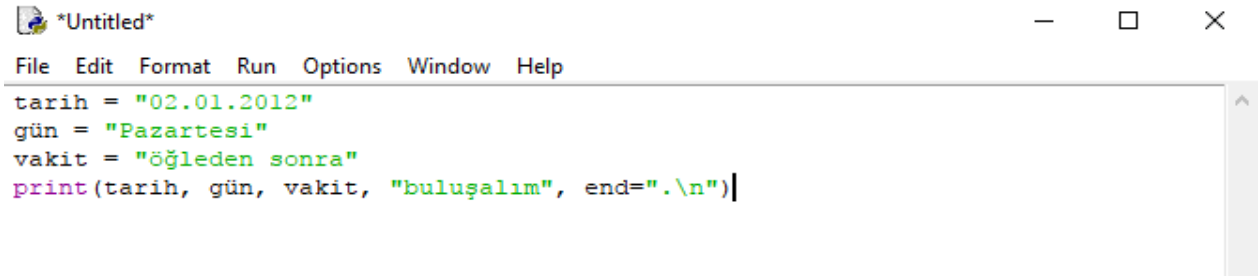
99) Unutmayın, aşağıdaki ekrana yazdığınız kodlar kapattığınızda kaybolur. IDLE'ı açtığınızda şöyle bir ekranla karşılaşacaksınız:

Sol üst köşede **File [Dosya]** menüsüne tıklayın ve menü içindeki **New Window [Yeni Pencere]** düğmesine basın. Aşağıdaki gibi bir ekranla karşılaşacaksınız. İşte Python kodlarımızı bu beyaz ekrana yazacağız.



100) Açtığımız yeni dosyaya aşağıdaki kodları yazalım. Dikkat ettiyseniz artık >>> sembolü yok.

```
tarih = "02.01.2012"
gün = "Pazartesi"
vakit = "öğleden sonra"
print(tarih, gün, vakit, "buluşalım", end=".\n")
```



Bu noktadan sonra **File > Save as** yolunu takip ederek programımızı masaüstüne **randevu.py** adıyla kaydediyoruz. Şu anda programımızı yazdık ve kaydettik. Artık programımızı çalıştırabiliriz. Bunun için **Run > Run Module** yolunu takip etmeniz veya kısaca **F5** tuşuna basmanız yeterli olacaktır. Bu iki yöntemden birini kullanarak programınızı çalıştırdığınızda şöyle bir çıktı elde edeceksiniz:

```
02.01.2012 Pazartesi öğleden sonra buluşalım.
```

Tebrikler! İlk Python programınızı yazıp çalıştırdınız.