Python Programlama Dili

1. Temel Bilgiler

1.1. Python Hakkında

Python, Guido Van Rossum adlı Hollandalı bir programcı tarafından yazılmış bir programlama dilidir. Geliştirilmesine 1990 yılında başlanan Python; C ve C++ gibi programlama dillerine kıyasla;

1. daha kolay öğrenilir,
2. program geliştirme sürecini kısaltır,
3. bu programlama dillerinin aksine ayrı bir derleyici programa ihtiyaç duymaz,
4. hem daha okunaklıdır, hem de daha temiz bir sözdizimine sahiptir.

Python'un bu ve buna benzer özellikleri sayesinde dünya çapında ün sahibi büyük kuruluşlar (Google, Yahoo! ve Dropbox gibi) bünyelerinde her zaman Python programcılarına ihtiyaç duyuyor.

1.2. Python nasıl kurulur?

Python'u kullanabilmek için, bu programlama dilinin sistemimizde kurulu olması gerekiyor. İşte biz de bu bölümde Python'u sistemimize nasıl kuracağımızı öğreneceğiz.

1.2.2. Microsoft Windows

Python'un resmi sitesindeki indirme adresinde (http://www.python.org/download) GNU/Linux kaynak kodlarıyla birlikte programın Microsoft Windows işletim sistemiyle uyumlu sürümlerini de bulabilirsiniz. Bu adresten Python'u indirmek isteyen çoğu Windows kullanıcısı için uygun sürüm Python 2.7.x Windows Installer (Windows binary – does not include source) olacaktır.

Eğer Python programlama dilinin hangi sürümünü kullanmanız gerektiği konusunda kararsızlık yaşıyorsanız, ben size 2.7 sürümlerinden herhangi birini kullanmanızı tavsiye ederim.

Windows sürümlerinin hiçbirinde Python kurulu olarak gelmez. O yüzden Windows 7 kullanıcıları, Python'ı sitesinden indirip kuracak. Bunun için şu adımları takip ediyoruz:

1. https://www.python.org adresini ziyaret ediyoruz.
2. Orada, üzerinde 'python-2.7.9.msi' yazan bağlantıya tıklıyoruz.
3. İnen dosyaya çift tıklayıp normal bir şekilde kuruluma başlıyoruz.
4. Kurulum adımlarından birinde şöyle bir ekranla karşılaşacaksınız:

5. Burada *Add python.exe to Path* (python.exe'yi yola ekle) diye bir seçenek görüyorsunuz. Tahmin edebileceğiniz gibi, bu seçenek Python programlama dilininin kurulu olduğu dizini YOL (*PATH*) dizinleri arasına ekleyerek, Python'ı kurulumdan sonra sadece adını kullanarak çalıştırabilmemizi sağlayacak.
6. Bu seçeneğin yanındaki küçük siyah oka tıklayarak, açılan menüden *Entire feature will be installed on local hard drive* girdisini seçiyoruz. Bundan sonra kuruluma normal bir şekilde devam edebiliriz.

1.4. print Komutu¶

Bir önceki bölümde Python'un komut satırına nasıl ulaşacağımızı görmüştük. (Bu komut satırına Pythonca'da "Etkileşimli Kabuk" veya "Yorumlayıcı" adı verilir.) Şimdi yukarıda anlattığımız yöntemlerden herhangi birini kullanarak Python'un etkileşimli kabuğunu açalım ve şuna benzer bir ekranla karşılaşalım:

```
Python 2.7.4 (default, Apr 10 2013, 12:11:55)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-54)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Komutlarımızı bu ">>>" işaretinden hemen sonra, hiç boşluk bırakmadan yazmaya başlayacağımızı daha önce söylemiştik.

Bu bölümde inceleyeceğimiz ilk komutumuzun adı print.

```
>>> print "Ben Python, Monty Python!"
```

Bu satırı yazıp ENTER tuşuna bastıktan sonra ekranda “Ben Python, Monty Python!” çıktısını görmemiz gerekiyor.

print komutu, Python’daki en önemli ve en temel komutlardan biridir. Python’la yazdığınız programlarda kullanıcılarınıza herhangi bir mesaj göstermek istediğinizde bu print komutundan yararlanacaksınız.

Burada biz istersek çift tırnak yerine tek tırnak (‘) da kullanabiliriz:

```
>>> print 'Ben Python, Monty Python!'
```

Parantez içinde de kullanabiliriz:

```
>>> print (“Ben Python, Monty Python!”)
```

Ancak karakter dizilerini tanımlarken, karakter dizisi içindeki başka kesme ve tırnak işaretlerine karşı dikkatli olmalıyız.

```
>>> print "Linux'un faydaları"
```

Bu komut bize hatasız bir şekilde “Linux’un faydaları” çıktısını verir. Ancak aynı işlemi tek tırnakla yapmaya çalışırsak şöyle bir hata mesajı alırız:

```
File "<stdin>", line 1
 print 'Linux'un faydaları'
 ^
SyntaxError: invalid syntax
```

Bunun nedeni, “Linux’un” kelimesindeki kesme işaretinden ötürü Python’un tırnakların nerede başlayıp nerede bittiğini anlamamasıdır. Eğer mutlaka tek tırnak kullanmak istiyorsak, kodu şu hale getirmemiz gerekir:

```
>>> print 'Linux\'un faydaları'
```

Aynı şekilde şu kodlar da hata verecektir:

```
>>> print "Ahmet, "Adana'ya gidiyorum," dedi."
```

Buradaki hatanın sebebi de, karakter dizisini başlatıp bitiren tırnaklarla, Ahmet'in sözünü aktarmamızı sağlayan tırnak işaretlerinin birbirine karışmasıdır.

Bu hatayı da şu şekilde önleyebiliriz:

```
>>> print "Ahmet, \"Adana'ya gidiyorum,\" dedi."
```

Buradaki "\" işaretleri olası bir hatadan kaçmamızı sağlar. Bu tür ifadelere Python dilinde Kaçış Dizileri (Escape Sequences) adı verilir.

1.5. Python'da Sayılar ve Aritmetik İşlemler

Python'da henüz dört dörtlük bir program yazamasak da en azından şimdilik onu basit bir hesap makinesi niyetine kullanabiliriz.

Örneğin:

```
>>> 2 + 5

7
```

... veya:

```
>>> 5 - 2

3
```

... ya da:

```
>>> 2 * 5

10
```

... hatta:

```
>>> 6 / 2

3
```

Bunların dışında, işimize yarayacak birkaç işleç daha öğrenelim:

```
>>> 2 ** 2

4
```

```
>>> 2 ** 3

8
```

Burada gördüğümüz ** işleci kuvvet hesaplama işlemleri için kullanılır. Mesela yukarıdaki iki örnekte sırasıyla 2 sayısının 2. ve 3. kuvvetlerini hesapladık.

Bir sayının 2. kuvveti o sayının karesidir. Aynı şekilde bir sayının 0.5. kuvveti de o sayının kareköküdür:

```
>>> 144 ** 0.5

12.0
```

% işleci ise bölme işleminden kalan sayıyı gösterir.

```
>>> 10 % 2

0
>>> 5 % 2

1
```

Gördüğünüz gibi sayıları yazarken tırnak işaretlerini kullanmıyoruz. Eğer tırnak işareti kullanırsak Python yazdıklarımızı sayı olarak değil karakter dizisi olarak algılayacaktır. Bu durumu birkaç örnekle görelim:

```
>>> 25 + 50

75
```

Bu komut, 25 ve 50'yi toplayıp sonucu çıktı olarak verir. Şimdi aşağıdaki örneğe bakalım:

```
>>> "25 + 50"

25 + 50
```

Bu komut 25 ile 50'yi toplamak yerine, ekrana "25 + 50" şeklinde bir çıktı verecektir. Peki, şöyle bir komut verirsek ne olur?

```
>>> "25" + "50"
```

Böyle bir komutla karşılaşan Python derhal “25” ve “50” karakter dizilerini (bu sayılar tırnak içinde olduğu için Python bunları sayı olarak algılamaz) yan yana getirip birleştirecektir. Yani şöyle bir şey yapacaktır:

```
>>> "25" + "50"

2550
```

Şimdi matematik işlemlerine geri dönelim. Öncelikle şu komutun çıktısını inceleyelim:

```
>>> 5 / 2

2
```

Ama biz biliyoruz ki 5’i 2’ye bölerseniz 2 değil 2,5 çıkar… Aynı komutu bir de şöyle deneyelim:

```
>>> 5.0 / 2

2.5
```

Gördüğünüz gibi bölme işlemini oluşturan bileşenlerden birinin yanına ”.0” koyulursa sorun çözülüyor. Böylelikle Python bizim sonucu tamsayı yerine kayan noktalı (floating point) sayı cinsinden görmek istediğimizi anlıyor. Bu ”.0” ifadesini istediğimiz sayının önüne koyabiliriz. Birkaç örnek görelim:

```
>>> 5 / 2.0

2.5

>>> 5.0 / 2.0

2.5
```

Python’da aritmetik işlemler yapılırken alıştığımız matematik kuralları geçerlidir. Yani mesela aynı anda bölme çıkarma, toplama, çarpma işlemleri yapılacaksa işlem öncelik sırası, önce bölme ve çarpma sonra toplama ve çıkarma şeklinde olacaktır. Örneğin:

```
>>> 2 + 6 / 3 * 5 - 4
```

işleminin sonucu “8” olacaktır. Tabii biz istersek parantezler yardımıyla Python’un kendiliğinden kullandığı öncelik sırasını değiştirebiliriz.

Buraya kadar Python’da üç tane veri tipi (data type) olduğunu gördük. Bunlar:

- Karakter dizileri (strings)
- Tamsayılar (integers)
- Kayan noktalı sayılar (floating point numbers)

1.6. Değişkenler

```
>>> n = 5
```

ifadesinde “n” bir değişkendir. Bu “n” değişkeni “5” verisini sonradan tekrar kullanılmak üzere depolar. Python komut satırında n = 5 şeklinde değişkeni tanımladıktan sonra n komutunu verirsek ekrana yazdırılacak veri 5 olacaktır. Yani:

```
>>> n = 5
>>> n

5
```

Bu “n” değişkenini alıp bununla aritmetik işlemler de yapabiliriz:

```
>>> n * 2

10

>>> n / 2.0

2.5
```

Şu örneklere bir göz atalım:

```
>>> a = 5
>>> b = 3
>>> a * b

15

>>> print "a ile b'yi çarparsak", a * b, "elde ederiz"

a ile b'yi çarparsak 15 elde ederiz
```

Burada değişkenleri karakter dizileri arasına nasıl yerleştirdiğimize, virgülleri nerede kullandığımıza dikkat edin.

Aynı değişkenlerle yaptığımız şu örneğe bakalım bir de:

```
>>> print a, "sayısı", b, "sayısından büyüktür"
```

Değişkenleri kullanmanın başka bir yolu da özel işaretler yardımıyla bunları karakter dizileri içine gömmektir. Şu örneğe bir bakalım:

```
>>> print "%s ile %s çarpılırsa %s elde edilir" %(a, b, a*b)
```

Burada, parantez içinde göstereceğimiz her bir öğe için karakter dizisi içine "%s" işaretini ekliyoruz. Karakter dizisini yazdıktan sonra da "%" işaretinin ardından parantez içinde bu işaretlere karşılık gelen değerleri teker teker tanımlıyoruz. Buna göre birinci değerimiz 'a' (yani 5), ikincisi 'b' (yani 3), üçüncüsü ise bunların çarpımı (yani 5 * 3)...

Bu yapıyı daha iyi anlayabilmek için bir iki örnek daha verelim:

```
>>> print "%s ve %s iyi bir ikilidir." %("Python", "Django")

>>> print "%s sayısının karesi %s sayısıdır." %(12, 12**2)

>>> print "Adım %s, soyadım %s" %("Fırat", "Özgül")
```

Gördüğünüz gibi, '%s' işaretleri ile hem değişkenleri hem de doğrudan değerleri kullanabiliyoruz. Ayrıca bu işaretler, bir karakter dizisi içine çeşitli değerleri kolaylıkla yerleştirmemizi de sağlıyor.

1.7. Python Scriptleri

Windows kullanıcıları IDLE adlı metin düzenleyici ile çalışabilirler. IDLE'a ulaşmak için Başlat/Programlar/Python/IDLE (Python GUI) yolunu takip ediyoruz. IDLE'ı çalıştırdığımızda gördüğümüz >>> işaretinden de anlayabileceğimiz gibi, bu ekran aslında Python'un etkileşimli kabuğudur.

Burada File menüsü içindeki New Window düğmesine tıklayarak boş bir sayfa açıyoruz.

İşte Python kodlarını yazacağımız yer burası. Şimdi bu boş sayfaya şu kodları ekliyoruz:

```
a = "elma"
b = "armut"
c = "muz"

print "bir", a, "bir", b, "bir de", c, "almak istiyorum"
```

Kodlarımızı yazdıktan sonra yapmamız gereken şey dosyayı bir yere kaydetmek olacaktır. Bunun için File/Save as yolunu takip ederek dosyayı deneme.py adıyla masaüstüne kaydediyoruz.

Dosyayı kaydettikten sonra **Run/Run Module** yolunu takip ederek veya doğrudan F5 tuşuna basarak yazdığımız programı çalıştırabiliriz.

Eğer programınızı IDLE üzerinden değil de, doğrudan MS-DOS komut satırını kullanarak çalıştırmak isterseniz şu işlemleri yapın:

Başlat/Çalıştır yolunu takip edip, açılan pencereye “cmd” yazın ve ENTER tuşuna basın.

Şu komutu vererek, masaüstüne, yani dosyayı kaydettiğiniz yere gelin:

```
cd C:/Documents and Settings/Kullanici_adi/Desktop
```

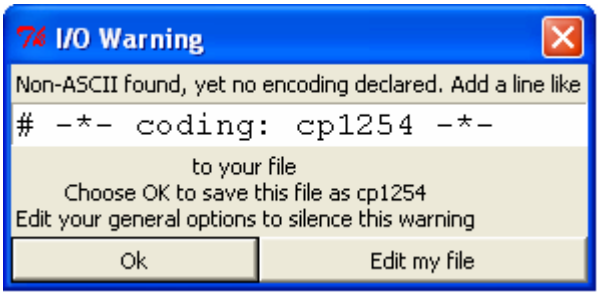
Masaüstüne geldikten sonra şu komutu vererek programınızı çalıştırabilirsiniz:

```
python deneme.py
```

1.8. Türkçe Karakter Sorunu¶

Python bazı durumlarda Türkçe karakter problemi çıkartabilmektedir.

Eğer IDLE üzerinde çalışıyorsanız programınızı herhangi bir dil kodlaması belirtmeden kaydetmeye çalıştığınızda şöyle bir uyarı penceresiyle karşılaşabilirsiniz:



Burada IDLE, dil kodlamasını belirtmeniz konusunda sizi uyarıyor. Eğer bu ekranda “Edit My File” düğmesine basacak olursanız, IDLE programınızın ilk satırına sizin yerinize `# -*- coding: cp1254 -*-` komutunu ekleyecektir...

1.9. Kullanıcıyla İletişim: Veri Alış-Verişi

Python'da kullanıcıdan birtakım veriler alabilmek, yani kullanıcıyla iletişime geçebilmek için iki tane fonksiyondan faydalanılır.

1.9.1. raw_input() fonksiyonu¶

raw_input() fonksiyonu kullanıcılarımızın veri girmesine imkân tanır.

```
raw_input("Lütfen parolanızı girin:")
print "Teşekkürler!"
```

Şimdi bu belgeyi "deneme.py" ismiyle kaydediyoruz. Daha sonra bir konsol ekranı açıp, programımızın kayıtlı olduğu dizine geçerek şu komutla programımızı çalıştırıyoruz:

```
python deneme.py
```

İsterseniz şimdi yazdığımız bu programı biraz geliştirelim. Mesela programımız şu işlemleri yapsın:

- Program ilk çalıştırıldığında kullanıcıya parola sorsun,
- Kullanıcı parolasını girdikten sonra programımız kullanıcıya teşekkür etsin,
- Bir sonraki satırda kullanıcı tarafından girilen bu parola ekrana yazdırılsın,
- Kullanıcı daha sonraki satırda, parolanın yanlış olduğu konusunda uyarılsın.

```
# -*- coding: cp1254 -*-
parola = raw_input("Lütfen parolanızı girin:")
print "Teşekkürler!"
print "Girdiğiniz parola: ", parola
print "Ne yazık ki doğru parola", parola, "değil."
```

Bu "parola" değişkenini karakter dizisi içine eklemenin başka bir yolu da kodu şu şekilde yazmaktır:

```
print "Ne yazık ki doğru parola %s değil" %(parola)
```

1.9.2. input() fonksiyonu¶

Tıpkı raw_input() fonksiyonunda olduğu gibi, input() fonksiyonuyla da kullanıcılardan bazı bilgileri alabiliyoruz.

```
# -*- coding: cp1254 -*-
a = input("Lütfen bir sayı girin:")
b = input("Lütfen başka bir sayı daha girin:")
print a + b
```

Kullanım açısından, görüldüğü gibi, raw_input() ve input() fonksiyonları birbirlerine çok benzer. Ama bunların arasında çok önemli bir fark vardır. Hemen yukarıda verilen kodları bir de raw_input() fonksiyonuyla yazmayı denersek bu fark çok açık bir şekilde ortaya çıkacaktır:

```
# -*- coding: cp1254 -*-
a = raw_input("Lütfen bir sayı girin:")
b = raw_input("Lütfen başka bir sayı daha girin:")
print a + b
```

Bu kodları yazarken input() fonksiyonunu kullanırsak, kullanıcı tarafından girilen sayılar birbirleriyle toplanacaktır. Diyelim ki ilk girilen sayı “25”, ikinci sayı ise “40” olsun. Programın sonunda elde edeceğimiz sayı “65” olacaktır.

Ancak bu kodları yazarken eğer raw_input() fonksiyonunu kullanırsak, girilen sayılar birbirleriyle toplanmayacak, sadece yan yana yazılacaklardır. Yani elde edeceğimiz şey “2540” olacaktır.

- **raw_input() fonksiyonu kullanıcının girdiği verileri karakter dizisine dönüştürür**.
- **input() fonksiyonu kullanıcıdan gelen verileri olduğu gibi alır.** Yani bu verileri karakter dizisine dönüştürmez. Bu yüzden, eğer kullanıcı bir sayı girmişse, input() fonksiyonu bu sayıyı olduğu gibi alacağı için, bizim bu sayıyla aritmetik işlem yapmamıza müsaade eder.

Bu durumu daha iyi anlayabilmek için mesela aşağıda raw_input() fonksiyonuyla yazdığımız kodları siz bir de input() fonksiyonuyla yazmayı deneyin:

```
# -*- coding: cp1254 -*-
isim = input("isminiz: ")
soyisim = input("soyisminiz: ")
print isim, soyisim
```

Eğer bu kodları input() fonksiyonuyla yazmayı denediyseniz, Python’un ilk veri girişinden sonra şöyle bir hata verdiğini görmüşsünüzdür:

```
SyntaxError: invalid syntax
```

Burada hata almamak için şöyle yapmak gerek:

```
>>> "Ahmet"

'Ahmet'
```

Dolayısıyla Python'un input() fonksiyonuyla bu hatayı vermemesi için de tek yol, kullanıcının ismini ve soyismini tırnak içinde yazması olacaktır.

- eğer biz programımız aracılığıyla kullanıcılardan bazı sayılar isteyeceksek ve eğer biz bu sayıları işleme sokacaksak (çıkarma, toplama, bölme gibi...) input() fonksiyonunu tercih edebiliriz.
- eğer biz kullanıcılardan sayı değil de karakter dizisi girmesini istiyorsak raw_input() fonksiyonunu kullanacağız.

1.10. Dönüştürme İşlemleri

Pek çok durumda bir sayıyı karakter dizisine ve eğer mümkünse bir karakter dizisini de sayıya dönüştürmek zorunda kalacaksınız. Şimdi dilerseniz bu duruma çok basit bir örnek verelim.

```
>>> a = "23"
```

Bildiğiniz gibi yukarıdaki a değişkeni bir karakter dizisidir. Şimdi bunu sayıya çevirelim:

```
>>> b=int(a)
>>> b+3

26
```

Böylece "23" karakter dizisini sayıya çevirmiş olduk. Ancak tahmin edebileceğiniz gibi her karakter dizisi sayıya çevrilemez. int() fonksiyonu yalnızca sayı değerli karakter dizilerini sayıya dönüştürebilir:

```
>>> kardiz = "elma"

>>> int(kardiz)

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'elma'
```

Gördüğünüz gibi, “elma” karakter dizisi sayı olarak temsil edilemeyeceği için Python bize bir hata mesajı gösteriyor.

```
# -*- coding: cp1254 -*-
ilk_sayi = int(raw_input("İlk sayıyı girin: "))
ikinci_sayi = int(raw_input("İkinci sayıyı girin: "))
toplam = ilk_sayi + ikinci_sayi
print "Bu iki sayının toplamı: ", toplam
```

Gördüğünüz gibi, burada yaptığımız şey çok basit. raw_input() fonksiyonunu tümden int() fonksiyonu içine aldık:

```
int(raw_input("İlk sayıyı girin: "))
```

Artık raw_input() fonksiyonuyla da aritmetik işlemler yapabiliyoruz.

```
# -*- coding: cp1254 -*-
sayi = int(raw_input("Bir sayı girin. Ben size bu sayının "
"istediğiniz kuvvetini hesaplayayım: "))

kuvvet = int(raw_input("Şimdi de %s sayısının kaçıncı kuvvetini "
"hesaplamak istediğinizi söyleyin: " %sayi))

print "%s sayısının %s. kuvveti %s olur." %(sayi, kuvvet, sayi ** kuvvet)
```

Burada, yazdığımız kodların nasıl işlediğine dikkat etmenin yanısıra, kodları görünüş açısından nasıl düzenlediğimize ve satırları nasıl böldüğümüze de dikkat edin.

Peki, yukarıda yaptığımız şeyin tersi mümkün mü? Yani acaba bir sayıyı karakter dizisine çevirebilir miyiz? Bu sorunun yanıtı evettir. Bu işlem için de str() adlı fonksiyondan yararlanacağız:

```
>>> a = 23
>>> str(a)

'23'
```

Böylece 23 sayısını, bir karakter dizisi olan ‘23’e dönüştürmüş olduk.

Gördüğünüz gibi, int() ve str() adlı fonksiyonlar yardımıyla karakter dizileri ve tamsayılar arasında dönüştürme işlemi yapabiliyoruz. Eğer bir sayıyı veya sayı değerli karakter dizisini kayan noktalı sayıya dönüştürmek istersek de float() adlı fonksiyondan yararlanacağız:

```
>>> a = 23

```

```
>>> float(a)

23.0

>>> float("34")

34.0
```

1.11. Yorum Satırları

Python'da kodlar içine nasıl açıklama/yorum eklenir, biraz da bundan bahsedelim:

Python'da kod içine açıklayıcı notlar eklemek için "#" işaretini kullanıyoruz.

Hemen bir örnek verelim:

```
print "deneme 1, 2, 3" #deneme yapıyoruz...
```

1.12. İşleçler

Türkçede işleç yerine operatör, işlenen yerine de operand dendiğine tanık olabilirsiniz.

1.12.1. Aritmetik İşleçler

Önceki derslerimizde temel işleçlerin bazılarını öğrenmiştik. İsterseniz bunları şöyle bir hatırlayalım:

+	toplama
-	çıkarma
*	çarpma
/	bölme
**	kuvvet

+ ve * işleçleri Python'da birden fazla anlama gelir. Örneğin yukarıdaki örnekte + işleci, işlenenler arasında bir toplama ilişkisi kuruyor. Ama aşağıdaki durum biraz farklıdır:

```
>>> "maltepe" + ".edu"

'maltepe.edu'
```

Burada + işleci işlenenler arasında bir birleştirme ilişkisi kuruyor.

Tıpkı + işlecinde olduğu gibi, * işleci de Python'da birden fazla anlama gelir. Bu işlecin, çarpma ilişkisi kurma işlevi dışında tekrar etme ilişkisi kurma işlevi de vardır. Yani:

```
>>> "hızlı " * 2

'hızlı hızlı '
```

...veya:

```
>>> "-" * 30

'------------------------------'
```

Burada * işlecinin, sayılar arasında çarpma işlemi yapmak dışında bir görev üstlendiğini görüyoruz.

Yukarıda verilen dört adet temel aritmetik işlece şu aritmetik işleci de ekleyelim:

%	modülüs

Örnek:

```
>>> 30 % 4

2
```

Gördüğünüz gibi modülüs işleci (%) gerçekten de bölme işleminden kalan sayıyı gösteriyor. Mesela bu bilgiyi kullanarak bir sayının tek mi yoksa çift mi olduğunu tespit edebiliriz:

```
sayi = input("Bir sayı girin: ")

if (sayi % 2) == 0:
    print("Girdiğiniz sayı bir çift sayıdır.")
else:
    print("Girdiğiniz sayı bir tek sayıdır.")
```

Eğer bir sayı *2*'ye bölündüğünde kalan değer *0* ise o sayı çifttir. Aksi halde o sayı tektir.

Ayrıca bir sayının son basamağını elde etmek için de modülüsten yararlanabilirsiniz. Herhangi bir tamsayı *10*'a bölündüğünde kalan (yani modülüs), bölünen sayının son basamağı olacaktır:

```
>>> 543 % 10

3
```

Şimdiye kadar öğrendiğimiz ve yukarıdaki tabloda andığımız bir başka aritmetik işleç de kuvvet işleci (**) idi. Mesela bu işleci kullanarak bir sayının karesini hesaplayabileceğimizi biliyorsunuz:

```
>>> int(625 ** 0.5)

25
```

Kuvvet hesaplamaları için ** işlecinin yanısıra `pow()` adlı bir fonksiyondan da yararlanabileceğimizi öğrenmiştik:

```
>>> pow(25, 2)

625
```

1.12.2 Karşılaştırma İşleçleri

Adından da anlaşılacağı gibi, karşılaştırma işleçleri, işlenenler (*operands*) arasında bir karşılaştırma ilişkisi kuran işleçlerdir. Bu işleçleri şöyle sıralayabiliriz:

==	eşittir
!=	eşit değildir
>	büyüktür
<	küçüktür
>=	büyük eşittir
<=	küçük eşittir

Burada da bunlarla ilgili basit bir örnek vererek yolumuza devam edelim:

```
parola = "xyz05"
```

```
soru = raw_input("parolanız: ")

if soru == parola:
    print("doğru parola!")

elif soru != parola:
    print("yanlış parola!")
```

Yukarıdaki örnekte == (eşittir) ve *!=* (eşit değildir) işleçlerinin kullanımını örneklendirdik. Öteki karşılaştırma işleçlerinin de nasıl kullanıldığını biliyorsunuz. Basit bir örnek verelim:

```
sayi = input("sayı: ")

if int(sayi) <= 100:
    print("sayı 100 veya 100'den küçük")

elif int(sayi) >= 100:
    print("sayı 100 veya 100'den büyük")
```

1.12.3. Değer Atama İşleçleri

Bu noktaya kadar yaptığımız çalışmalarda sadece tek bir değer atama işleci gördük. Bu işleç = işlecidir. Mesela:

```
>>> a = 23
```

Python'daki tek değer atama işleci elbette = değildir. Bunun dışında başka değer atama işleçleri de bulunur.

+= işleci

Bu işlecin ne işe yaradığını anlamak için şöyle bir örnek düşünün:

```
>>> a += 5
>>> print(a)

28
```

-= işleci

Bir önceki += işleci toplama işlemi yapıp, ortaya çıkan değeri tekrar aynı değişkene atıyordu. -= işleci de buna benzer bir işlem gerçekleştirir:

```
>>> a = 23
```

```
>>> a -= 5
>>> print(a)

18
```

/= işleci

Bu işlecin çalışma mantığı da yukarıdaki işleçlerle aynıdır:

```
>>> a = 30
>>> a /= 3
>>> print(a)

10
```

***= işleci**

Bu da ötekiler gibi, çarpma işlemi yapıp, bu işlemin sonucunu aynı değişkene atar:

```
>>> a = 20
>>> a *= 2
>>> print(a)

40
```

%= işleci

Bu işlecimiz ise bölme işleminden kalan sayıyı aynı değişkene atar:

```
>>> a = 40
>>> a %= 3
>>> print(a)

1
```

****= işleci**

Bu işlecin ne yaptığını tahmin etmek zor değil. Bu işlecimiz, bir sayının kuvvetini hesapladıktan sonra çıkan değeri aynı değişkene atıyor:

```
>>> a = 12
>>> a **= 2
>>> print(a)

144
```

Eşdeğeri:

```
>>> a = 12
>>> a = a ** 2
>>> print(a)

144
```