Bölüm

1

Giriş

1.1 Başlarken

Neden Python?

Bu metinde bazı bilimsel hesaplamaları yapmak için Python programlama dilini kullanmayı öğreneceğiz. Python, *Guido von Rossum* tarafından geliştirilen ve ilk sürümü 1991 yılında yayınlanan bir programlama dilidir. Python oldukça yüksek seviyeli bir programlama dilidir, yani konuşma diline çok benzerdir ve öğrenilmesi çok kolaydır. Günümüzde bu programlama dilinin çok popüler olmasının bir çok sebebi vardır, bazılarını sıralayalım.

- Son derece basittir, kolay öğrenilir.
- Ücretsiz ve açık kaynak kodludur.
- Çok yüksek seviyeli bir dildir, kısa kod parçalarıyla karmaşık işlemler yapılabilir.
- Portatiftir, programlar her sistemde aynı şekilde çalışır.
- Geniş bir modül kütüphanesi vardır, bir çok amaç için önceden hazırlanmış kullanıma hazır modüller vardır.
- Geniş bir kullanıcı topluluğu vardır, destek ve dokümantasyon bulmak kolaydır.
- Kaynak kodundan gerçek zamanlı olarak çalışır, kodların derlenmeye ihtiyacı yoktur.
- Tamamen nesne yönelimlidir, büyük ölçekli ve kompleks programlar yazmak kolaydır.

Python programlama dilinin iki farklı verisyonu var; versiyon 2 ve versiyon 3. Versiyon 3 daha yeni olan ve gelecek güncellemelere daha açık olandır, fakat bilimsel hesaplamalar açısından versiyon 2 daha kullanışlıdır. Çünü bazı kütüphaneler hala versiyon 3 ile uyumlu değil, bundan dolayı biz versiyon 2'yi kullanacağız. Versiyonlar arasında söz dizimi açısından çok az fark var, zamanla bu farkları öğrenip kolayca versiyon 3'e geçebileceksiniz.

Python Yazılımı

Bu bölümde, Python ile programlamaya başlamadan önce öğrenmemiz gereken bazı önemli noktalara değineceğiz. Önce Python yazılımını bilgisayarımıza kurup kullanmayı öğreneceğiz, daha sonra da ilk programımızı yazıp çalıştıracağız.

Yapacağımız ilk iş Python yazılımını (Python **interpreter** denir) kişisel bilgisayarımıza kurmak olacak. Bu işlemi Linux ve Windows tabanlı işletim sistemlerinde benzer şekilde yaparız. Bilgisayarlar kullanıcılar tarafından girilen komutlar ile işlem yapar, grafiksel bir arayüzde fareniz ile bir butona tıkladığınızda arka planda bir komut girmiş olursunuz, bilgisayar da bu komut ile istenen işlemi yapar. Her bilgisayarda grafik arayüz olmadan doğrudan komutları girebileceğiniz bir yazılım bulunur, bu yazılım Mac OSX veya diğer Linux tabanlı işletim sistemlerinde genelde **Terminal**, Windows işletim sisteminde ise **PowerShell**'dir. Bilgisayarınız Mac OSX işletim sistemine sahip bir Mac bilgisayar ise Terminal uygulamasını çalıştırın ve açılan ekranda `python` yazıp enter tuşuna basın. Uygulama size bir mesaj ile cevap verecektir, bu mesajda sistemde yüklü Python versiyonu hakkında bilgi verir. Bütün Mac OSX işletim sistemleri içinde Python 2 verisyonu yüklü olarak gelir. Bundan dolayı böyle bir bilgisayarınız varsa otomatik olarak Python yazılımına sahipsiniz demektir ve yeni bir yükleme yapmanıza gerek yok, şimdi `exit()` komutunu girerek Python yazılımını sonlandırın. Aslında sistemde yüklü gelen Python yazılımı ile program geliştirmek pek önerilmez, bunun yerine yeni bir Python 2 yüklemesi yapmak iyi bir fikirdir. Ama eğitim amaçlı bunu kullanabiliriz, daha sonra isterseniz öğrenip yeni bir yükleme yapabilirsiniz.

Terminal

```
Terminal > python
Python 2.7.13 (default Apr 4 2017, 8:46:44)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more
↪  information.
```

Windows bilgisayarlarda PowerShell uygulamasını çalıştırıp `python` komutunu girerseniz muhtemelen bir hata mesajı ile karşılaşırsınız, çünkü sistemde Python yazılmı yüklü değildir. Yüklemek için Python web sitesine[1] gidip güncel Python yazılımını indirip bilgisayarınıza kurmanız gerekir, şu an güncel versiyon Python 2.7.15'tir. Şimdi tekrar `python` komutunu girin, yine bir hata mesajı alabilirsiniz. Bunun sebebi, yazılım sisteme kurulmuş olsa da komut istemcisinden `python` komutu ile çalışacak şekilde ayarlanmamış olabilir. Bunu sağlamak için PowerShell yazılımını tekrar çalıştırıp aşağıdaki komutu girin.

Terminal

```
[Environment]::SetEnvironmentVariable("Path",
↪  "$env:Path;C\Python27",
"User")
```

Artık PowerShell uygulamasını yeniden başlattığınızda Python yazılımını yukarıdaki gibi çalıştırıyor olmanız gerekiyor. `exit` komutuyla Python yazılımını kapatabilirsiniz.

İlk Program

Şimdi artık Python ile programlar yazıp çalıştırmaya hazırız. Bir Python programını çalıştırmanın farklı yolları vardır, en basiti kodları bir metin editöründe yazıp `program.py` şeklinde `.py` uzantılı bir dosya olarak kaydedip komut istemcisinde dosyanın bulunduğu klasöre gidip `python program.py` komutunu girmektir. Böylece komut istemcisinde programımız çalışmaya başlar. Komut istemcisinde `ls` komutunu girersek bulunduğumuz konumdaki

[1] https://www.python.org/downloads/

dosya ve klasörlerin listesini görürüz, burada listelenen bir klasöre girmek için `cd` komutunu kullanırız: `cd klasor1`. Bu şekilde konumumuzu öğrenip dosyayı kaydettiğimiz klasöre girebiliriz, kolaylık açısından programlarınızı bilgisayarınızın ana dizininde bir klasör oluşturup programlarınızı buraya kaydetmeniz önerilir. Şimdi ilk programımızı yazıp çalıştıralım. Bir programlama dili öğrenilirken yazılan ilk programın *hello world* programı olması adettendir, böyle bir programı Python ile yazmak oldukça kolay.

```
1 print "Hello, World."
```

Gördüğümüz gibi bu programı Python ile tek satırda yazabiliyoruz. Bu programı bir metin editöründe yazıp bilgisayarımızın ana klasörüne `hello.py` olarak kayededip çalıştırdığımızda ekranda `Hello, World!` mesajı görüntülenecektir. Burada ekrana yazdırma işini `print` ifadesi yapıyor.

Terminal

```
Terminal > python hello.py
Hello, World.
```

Bir Python programını çalıştırmanın diğer bir yolu da komut istemcisinde `python` komutunu girerek Python yazılımını çalıştırdıktan sonra programımızın kodlarını satır satır komut istemcisine girip enter tuşuna basmaktır, bundan sonra girdiğiniz kodun karşılığı olan çıktı anında ekranda görülür. Bu yolla bir dosyaya kodları kaydetmemize gerek yoktur fakat yukarıdaki gibi tek satırdan oluşan bir programımız yoksa bu metod kullanışsızdır, dosyayı kaydederek çalıştırdığınızda tek tek değil tüm dosya içeriği aynı anda çalıştırılır. Öğrenme amaçlı deneme yapmak için kullanırız genelde bu yöntemi, örnek olarak yukarıdaki programı bu yolla çalıştıralım. Önce `python` komutunu girerek bir Python **oturumu** başlatalım, sonra da komutumuzu girerek çıktısını alalım. Bu şekilde Python oturumu ile yapılan hesaplamalara **interaktif** hesaplama deriz.

Terminal

```
Terminal > python
Python 2.7.13 (default Apr 4 2017, 8:46:44)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more
 ↪  information.
>>> print "Hello, World"
Hello, World.
```

Başka bir yöntem de bir **IDE** (integrated developement environment) kullanmaktır. Bir IDE, hem metin editörünü hem de komut istemcisini aynı grafik arayüz içinde sunan bir yazılımdır. Bir çok kullanışlı IDE yazılımı vardır, ilgili okur araştırıp herhangi birini kullanabilir.

1.2 Değişkenler ve Basit Hesaplamalar

Basit Bir Hesaplama

Yerden v_0 ilk hızla havaya atılan bir topun bir t anındaki düşey konumunu veren $y(t)$ fonksiyonu

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

ile verilir ($g \approx 9.81$ m/sn^2 yerçekimi sabiti). Şimdi Python programlama dili yardımı ile bilgisayarı bir hesap makinesi gibi kullanarak bu formülü v_0 ve t'nin farklı değerleri için hesaplatacağız. Örnek olarak $v_0 = 10$ m/sn^2 ilk hızla fırlatılan nesnenin $t = 0.3$ sn zamanındaki konumunu hesaplatıp ekrana yazdıran bir program yazalım, Bunu Python da tek satırlık bir kod ile aşağıdaki gibi yapabiliriz.

```
1 print 10*0.3 - 0.5*9.81*0.3**2
```

Gördüğünüz gibi yukarıdaki formülde v_0, t ve g değerlerine yerine yazdık ve `print` ifadesiyle bunu ekrana yazdırdık. Python burada ekrana belirtilen işlemin sonucunu doğrudan yazdırır. Bu kodları `hesap1.py` olarak kaydedip çalıştırırsak aşağıdaki çıktıyı alırız.

Terminal

```
Terminal > python hesap1.py
2.55855
```

Aynı hesabı $v_0 = 2$ ve $t = 0.1$ için yapalım, `hesap2.py` olarak kaydedip çalıştıralım.

```
1 print 2*0.1 - 0.5*9.81*0.1**2
```

Çıktısı aşağıdaki gibi olcaktır.

Terminal

```
Terminal > python hesap2.py
0.15095
```

Şimdi bu yazdığımız programı analiz edelim. Burada kullandığımız ilk ifade `print`, bu ifade kendinden sonra gelen nesneyi ekrana yazdırmaya yarar. Daha sonra da ekrana yazdırılmasını istediğimiz ifadeyi Python dilinde yazdık. Cebirsel işlemler Python'da yazılırken günlük matematik yazımına benzer bir söz dizimi uygularız. Toplama, çıkarma, çarpma, bölme ve kuvvet alma işlemlerini sırasıyla `+`, `-`, `*`, `/`, `**` sembolleriyle kodlarız. Burada $(1/2) \cdot 9.81 \cdot 0.1^2$ sayısını kodlarken `1/2` sayısını `0.5` olarak yazmamızın özel bir sebebi var, az sonra değineceğiz. Python ile kodlama yaparken cebirsel işlem önceliği matematikte olduğu gibidir. Ayrıca bu temel işlemler dışında işlemler yapmamızı sağlayan farklı cebirsel işlem operatörleri de vardır, zamanı geldikçe öğreneceğiz. Bu programda boşlukların kullanımına da değinelim, - işaretinin boşluklarla ayrıldığı dikkatinizi çekmiş olabilir. Aynı satırda olduktan sonra boşlukların sayısı Python için fark oluşturmaz, `print 2*0.1-0.5*9.81*0.1**2` komutu da aynı sonucu verir. Fakat okunaklı olması açısından programımıza gerekli yerlerde boşluklar bırakabiliriz, matematiksel işlemlerde + ve - işaretlerini boşluklarla ayırmak, diğer işlemlerde boşluk kullanmamak işlemi daha kolay okunur yapacaktır.

Python Değişkenleri

Bir program yazarken genelde tüm sayıları yukarıda yaptığımız gibi işleme sokarak sonuca gitmeyiz, bunlara birer isim verip birer **değişken** haline getirip formülde isimleriyle kullanırız.

```
1 v0 = 2
2 g = 9.81
3 t = 0.1
4 y = v0*t - 0.5*g*t**2
5 print y
```

Yukarıdaki programda `v0`, `g`, `t` ve `y` isimli dört tane yeni değişken tanımladık. Bunlardan ilk üçüne birer değer atadık, dördüncüsünün de değerini Python'a nasıl hesaplayacağını tarif ettik. Daha sonra bu son değişkenin değerini ekrana yazdırdık. Python'da yeni bir değişken tanımlama işlemi `degisken_ismi = deger` komutuyla yapılır. Bu programı kaydedip çalıştırırsak öncekiyle aynı çıktıyı alırız, bu program sadece daha okunaklı. Yazacağımız programların olabildiğince okunaklı olmasını sağlamalıyız, bu sayede daha sonra mevcut hataları düzeltmek veya programı geliştirmek için yapacağımız güncellemeler yapmak daha kolay olacaktır. Değişkenleri isimlendirirken olabildiğince kısa ve amacına uygun isimler seçmeliyiz, bu da programın okunaklı olmasına katkıda bulunacaktır. Yukarıdaki programı şimdi de bir interaktif Python oturumunda çalıştıralım.

Terminal

```
Terminal > python
Python 2.7.13 (default Apr 4 2017, 8:46:44)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more
 ↪  information.
>>> v0 = 2
>>> g = 9.81
>>> t = 0.1
>>> y = v0*t - 0.5*g*t**2
>>> print y
0.15095
```

Değişken tanımlarken dikkat etmemiz gereken önemli bir nokta var. Bazı kelimeler Python tarafından rezerve edilmiştir ve programlarımızda bu kelimeleri değişkenlere isim olarak atayamayız. Bu kelimeler `and`, `as`, `assert`, `break`, `class`, `continue`, `def`, `del`, `elif`, `else`, `except`, `False`, `finally`, `for`, `from`, `global`, `if`, `import`, `in`, `is`, `lambda`, `None`, `nonlocal`, `not`, `or`, `pass`, `raise`, `return`, `True`, `try`, `with`, `while`, `yield` kelimeleridir. Bunların Python için özel anlamları vardır ve bu kelimeleri görünce Python bu anlamlarına göre hareket eder, bunların anlamlarını karşılaştıkça öğreneceğiz.

Yorumlar ve Metin Biçimleyiciler

Şimdi yukarıdaki programı daha okunaklı yapmak için kodların içine bazı notlar ekleyelim.

```
1 v0 = 2                  #baslangic hizi
2 g = 9.81                #yercekimi ivmesi
```

```
3  t = 0.1                 #zaman degiskeni
4  y = v0*t - 0.5*g*t**2   #konum hesaplamasi
5  print y
```

Python # karakteri ile karşılaştığında o satırın geri kalanını değerlendirmeden bir alt satıra atlar. Bu sayede programımıza bazı **yorumlar** ekleyebiliriz. Bu notlar program çıktısına etki etmez, sadece kodları daha okunaklı yapar. Burada yorumları yazarken Türkçe karakterler kullanmadığımız dikkatinizi çekmiştir, Python varsayılan olarak ASCII karakterlerle çalışır, eğer farklı bir karakter seti kullanmak istiyorsak bunu belirtmeliyiz. Örneğin Türkçe karakterler için UTF-8 karakterlere ihtiyacımız var bunun için programın ilk satırına `# -*- coding: utf-8 -*-` yazmalıyız. Bu satır # karakteriyle başlasa da Python tarafından yorum olarak kabul edilmez ve değerlendirilir. Ayrıca bu satırın programımızın ilk satırı olması zorunludur. Bu şekilde Türkçe karakterler sorunsuz bir şekilde kullanılabilir olsa da yukarıdaki örnekte yaptığımız gibi programımızı tamamen ASCII karakterlerle yazmanız önerilir. Bu metinde bundan sonra türkçe kelimeler sadece ASCII karakterler kullanılarak yazılacaktır.

```
1  # -*- coding: utf-8 -*-
2  v0 = 2                  #baslangıç hızı
3  g = 9.81                #yerçekimi ivmesi
4  t = 0.1                 #zaman değişkeni
5  y = v0*t - 0.5*g*t**2   #konum hesaplaması
6  print y
```

Programımızın kodlarını yeterince okunaklı yapmak sadece programcının yararınadır, kullanıcı için de programın çıktısını daha anlaşılır yapmalıyız. Şu anda programımız sadece ekrana bir sayı yazdırıyor, bu sayının anlamlı bir cümle içinde yazılması daha uygun olacaktır. Bunun için en sık kullanılan yöntem **printf** yöntemi olarak adlandırılır ve aşağıdaki gibi kullanılır.

```
1  v0 = 2
2  g = 9.81
3  t = 0.1
4  y = v0*t - 0.5*g*t**2
5  print "t=%g sn zamaninda topun yuksekligi %5.2f metredir." % (t, y)
```

Bu programı çalıştırırsanız çıktısı aşağıdaki gibi olacaktır.

Terminal

```
Terminal > python hesap6.py
t=0.1 sn zamaninda topun yerden yuksekligi 0.15 metredir.
```

Bu çıktı kullanıcı açısından yeteri kadar açıktır. Şimdi bu programda `print` ifadesinde metni nasıl biçimlendirdiğimizi açıklayalım. Bu yöntemde önce ekrana yazdıracağımız metni hazırlarız ve değişkenlerin yerleşeceği bölümleri belirleriz. Bu örnekte ekrana yazdırmak istediğimiz metin `"t=X sn zamaninda topun yerden yuksekligi Y metredir."` biçimindedir ve `X` ile `Y` kısımlarına bazı değişkenlerin değerlerinin yazılmasını istiyoruz. Python'da bu şekilde çift tırnak işareti içinde yazılan her şey bir saf metin olarak algılanır, bu tür nesnelere

string nesneleri denir. Nesne konusunu daha sonra detaylı olarak ele alacağız ama birkaç söz söyleyelim şimdi. Python'da herşey bir nesnedir, metinler, sayılar, formüller, fonksiyonlar, vb.. Bu bölümde metinlerin dışında sıklıkla kullandığımız tamsayılar **int** nesneleridir (integer), ondalıklı sayılar ise **float** nesneleridir. String nesneleri tek tırnak içinde de yazılabilir, `'t=X sn zamaninda topun yerden yuksekligi Y metredir.'` ifadesi de aynı anlamdadır. Şimdi bu string nesnesini biçimlendireceğiz, yani `X` ve `Y` yazan yerlere sırasıyla programımızda tanımladığımız `t` ve `y` değişkenlerinin değerlerini yazdıracağız. Bunun için bu karakterlerin olduğu yere `%` sembolüyle başlayan bir **biçimleyici** yerleştiririz. `%` karakterinden sonra buraya yerleştirilecek verinin formatını belirten kısaltmalar kullanırız. `%s` ile buraya bir string formatında, `%d` ile tamsayı formatında ve `%f` ile de 6 ondalık basmaklı olmak üzere bir float veri yazılacağı belirtilir. Eğer `%.2f` biçiminde girersek virgülden sonra iki ondalık basamaklı biçimlendirmesini istemiş oluruz, benzer şekilde `%5.2f` yazarsak virgülden sonra iki basamak olmak üzere toplam beş karakterlik bir biçimlendirme yaptırırız. Bilimsel formatta biçimlendirme çeşitleri de `%e` ve `%E` biçimindedir. `%g` veya `%G` ile mümkün olan en kompakt biçimde sayıları biçimlendiririz. Biçimleyicileri yerleştirdikten sonra metnin sonuna tekrar `%` işareti koyup bu biçimleyicilerin alacakları değişkenleri **sırasıyla** belirtiriz, tek bir değişken kullandıysak parantez kullanmadan da yazabiliriz. Eğer birden fazla satır kaplayan bir metin yazdırmak istersek bunu üç tane tırnak işareti arasında yazarız.

```
1  v0 = 2
2  g = 9.81
3  t = 0.1
4  y = v0*t - 0.5*g*t**2
5  print """
6  v0=%f m/sn ilk hizla firlatilan bir topun
7  t=%g sn zamaninda yerden yuksekligi %.2e metredir.
8  g=%3.2f m/sn^2 alinmistir.
9  """ % (v0, t, y, g)
```

Bu programın çıktısı aşağıdaki gibi olacaktır.

Terminal

```
Terminal > python hesap7.py

v0=2.000000 m/sn ilk hizla firlatilan bir topun
t=0.1 sn zamaninda yerden yuksekligi 1.51e-1 metredir.
g=9.81 m/sn^2 alinmistir.
```

Tam Sayı Bölmesi

Şimdi bu programda kullandığımız fomülü yeniden ele alalım. Bu formülü `y = v0*t - 0.5*g*t**2` yerine `y = v0*t - (1/2)*g*t**2` biçiminde değiştirelim.

```
1  v0 = 2
2  g = 9.81
3  t = 0.1
4  y = v0*t - (1/2)*g*t**2
5  print y
```

Programı çalıştırırsak beklemediğimiz bir çıktı alırız.

Terminal

```
Terminal > python hesap8.py
0.2
```

Gördüğünüz gibi yanlış hesaplama yapıldı. Bunun sebebi şudur, Python 2'de tamsayı nesneleri bölme işlemine alınınca sonuç yine bir tamsayı olarak döndürülür. Dolayısıyla bölme işlemi sonucu ondalıklı oluyorsa virgülden sonraki kısmı silinerek bu sayı bir integer nesnesine döndürülür. Bu örnekte parantez içinde bulunan `1/2` işlemi `0.5` değerine eşit olduğu için virgülden sonra gelen kısım silinince `0` tam sayısı elde edilir, hatanın kaynağı budur. Bu hatanın önüne geçmenin birkaç yolu vardır. İlk yöntem programımızın ilk satırına `from __future__ import division` satırını eklemektir. Bundan sonra programdaki tüm bölmeler tamsayı bölmesi yerine ondalıklı bölme işlemi olacaktır.

```
1 from __future__ import division
2 v0 = 2
3 g = 9.81
4 t = 0.1
5 y = v0*t - (1/2)*g*t**2
6 print y
```

Şimdi tekrar kontrol edersek işlemin hatasız yapıldığını görürüz.

Terminal

```
Terminal > python hesap9.py
0.15095
```

Tam sayı bölmesini engellemenin bir diğer yolu da programı çalıştırırken komut istemcisinde `-Qnew` parametresini girmektir.

Terminal

```
Terminal > python -Qnew hesap8.py
0.15095
```

Tam sayı bölmesini engellemenin önerilen yöntemi ise şudur, programımızda float bölmesi yapacaksak kullanacağımız sayıları birer float nesnesine çevirmeliyiz. Python'da bir float ve bir int nesnesinin toplamı, farkı, çarpımı ve bölümü her zaman bir float sonuç üretir, dolayısıyla böleceğimiz sayılardan sadece bir tanesini float yaparsak sorunu çözeriz. Bir int nesnesini float nesnesine çevirmenin en basit yolu sayıyı virgülden sonraki kısmı 0 olacak şekilde ondalıklı yazmaktır. Yani programımızda `1` yerine `1.0` yazarsak problem çözülür.

```
1 v0 = 2
2 g = 9.81
3 t = 0.1
4 y = v0*t - (1.0/2)*g*t**2
5 print y
```

Bu programı çalıştırırsak yine doğru hesaplama yapıldığını görürüz. Float dönüşümünün bir diğer yolu da `float()` fonksiyonunu kullanmaktır, bu fonksiyon aldığı argümanı mümkünse bir float nesnesine çevirir. Dolayısıyla aşağıdaki program da hatasız işlem yapacaktır.

```
v0 = 2
g = 9.81
t = 0.1
y = v0*t - (1/float(2))*g*t**2
print y
```

1.3 Matematiksel Fonksiyonlar

Şimdi biraz daha karmaşık hesaplamalar yapalım. Varsayalım ki bir top belirli bir ilk hızla yerden fırlatılıyor ve topun verilen bir $y = y_1$ konumuna ne kadar zamanda ulaşacağını hesaplamamız isteniyor. Bu problemi çözmek için

$$y_1 = v_0 t - \frac{1}{2} g t^2$$

formülünde t değişkenini çözmemiz gerekiyor. Bunun için denklemi

$$\frac{1}{2} g t^2 - v_0 t + y_1 = 0$$

biçiminde yazalım, bu denklem t'nin ikinci dereceden cebirsel bir denklemidir ve çözümü

$$t_1 = \frac{v_0 - \sqrt{v_0^2 - 2 g y_1}}{g} \quad \text{ve} \quad t_2 = \frac{v_0 + \sqrt{v_0^2 - 2 g y_1}}{g}$$

biçimindedir. Bu iki çözümden birisi topun yükselirken diğeri de geri düşerken y_1 konumuna ulaştığı zamanı belirtir. Bu hesaplamayı Python'da yapmak için $\sqrt{x}$ işlemine ihtiyacımız olacak.

Math Modülü

Python'da özel amaçlı fonksiyonlar, adına **modül** denilen dosyalarda saklanır ve bu fonksiyonları kullanmak için ilgili modülü kullanacağımızı programımızın içinde belirtmemiz gerekir. Python'da $\sqrt{x}$, $\sin(x)$, $\cos(x)$, $\log(x)$ gibi matematiksel hesaplamalar `math` modülü içinde tanımlıdır. $\sqrt{x}$ işlemi örneğin `math` modülü içinde tanımlı olan `sqrt()` fonksiyonu yardımıyla yapılır. Bir modül içindeki bir fonksiyonu kullanmadan önce programımızda `from modul import fonksiyon` komutunu girmemiz gerekiyor, bu işleme modülden ilgili fonksiyonu ithal etme (**import** etme) denir. Dolayısıyla yukarıdaki hesaplamayı aşağıdaki program ile yapabiliriz.

```
from math import sqrt

v0 = 5
g = 9.81
y1 = 0.2

t1 = (v0 - sqrt(v0**2 - 2*g*y1))/g
```

```
8  t2 = (v0 + sqrt(v0**2 - 2*g*y1))/g
9
10 print t1, t2
```

Bu programın çıktısı şu şekilde olacaktır.

Terminal

```
Terminal > python hesap12.py
0.0417063724983 0.977661619347
```

Burada `print` ifadesinden sonra iki veriyi virgülle ayırarak yazdık, bu şekilde virgülle ayrılarak sıralanmış nesneler aynı satırda yazdırılır. Ayrıca programın okunaklı olması açısından genellikle import ifadeleri programın ilk satırlarında yazılır ve gerektiğinde boş satırlar bırakılır. Programımızda aynı modülden birden fazla fonksiyon kullanılacaksa bunları virgülle ayırarak `from math import sqrt, sin, log, exp` biçiminde belirtebiliriz. Modüldeki tüm fonksiyonlara ihtiyacımız olacaksa `from math import *` komutu ile modüldeki tüm fonksiyonları programımızda kullanılabilir hale getirmiş oluruz. Bunun yerine sadece `import math` komutunu girersek de modülün tüm fonksiyonlarına erişim sağlamış oluruz, fakat bu durumda o fonksiyonları kullanırken modülün ismini ön ek olarak `math.sqrt()` biçiminde belirtmemiz gerekir.

```
1  import math
2
3  v0 = 5
4  g = 9.81
5  y1 = 0.2
6
7  t1 = (v0 - math.sqrt(v0**2 - 2*g*y1))/g
8  t2 = (v0 + math.sqrt(v0**2 - 2*g*y1))/g
9
10 print t1, t2
```

Burada kullandığımız ön eki istersek değiştirebiliriz. `import math as m` komutuyla fonksiyonları `m.sqrt()` gibi kullanabiliriz.

```
1  import math as m
2
3  v0 = 5
4  g = 9.81
5  y1 = 0.2
6
7  t1 = (v0 - m.sqrt(v0**2 - 2*g*y1))/g
8  t2 = (v0 + m.sqrt(v0**2 - 2*g*y1))/g
9
10 print t1, t2
```

`math` modülü içinde yukarıda bahsettiğimiz gibi matematiksel fonksiyonların yanında π ve e gibi bazı matematiksel sabitler de vardır.

```
1 from math import sin, cos, log, pi
2 x = 3*pi/2
3 s = sin(x-1)*cos(x+1) + log(2*x)
4 print s
```

Yukarıdaki program aşağıdaki çıktıyı üretecektir.

Terminal

```
Terminal > python hesap15.py
1.7886934611
```

Kompleks Sayılar

İkinci dereceden cebirsel denklemlerin kompleks kökleri var olabilir, bu durumla hesaplamalarda sıklıkla karşılaşırız ve dolayısıyla programlarımızda kompleks sayılarla işlem yapmaya ihtiyacımız olabilir. Şimdi biraz da Python'da kompleks sayı işlemlerine değinelim. Bir kompleks sayı, a ve b reel sayılar olmak üzere $a+i\cdot b$ biçiminde ifade edilir. Buradaki i sayısına **sanal birim** denir ve $i^2=-1$ yani $i=\sqrt{-1}$ eşitliği ile tanımlanır. Bu sayılar yardımıyla diskriminantı negatif olan $x^2+2=0$ gibi denklemlerin çözümleri $x_{1,2}=\pm\sqrt{2}i$ biçiminde ifade edilebilir. Python'da sanal birim `j` sembolü ile gösterilir, örneğin $1+2i$ kompleks sayısı `1 + 2j` biçiminde kodlanır. `2j` kısmını kodlarken arada `*` çarpma işareti kullanmadığımıza dikkat edin, ayrıca Python'da pür sanal i sayısını da `1j` olarak kodlarız. `j` sembolünün önünde mutlaka bir sayı gelmeli ve arada çarpma işareti bulunmamalıdır. Bu şekilde tanımlanmış değişkenler Python'da bir `complex` nesnesi olarak adlandırılır, Pythonda herhangi bir `x` nesnesinin hangi türden olduğunu anlamak için `type(x)` komutunu kullanırız. Bunları bir Python oturumunda gözlemleyelim.

Terminal

```
>>> x = 3-2j
>>> type(x)
<type 'complex'>
>>> y = 2-1j
>>> type(y)
<type 'complex'>
```

Kompleks sayı nesneleriyle işlem yaparken Python bunların kompleks sayı olduğunu algılar ve otomatik olarak kompleks sayı aritmetiğini kullanır. Şimdi yukarıda tanımladığımız sayılarla bazı işlemler yapalım.

Terminal

```
>>> x + y
(5-3j)
>>> 2*y
(4-2j)
>>> y + 2j
(2+1j)
```

Python'da kompleks sayıların girdiği işlemlerin sonucu her zaman bir `complex` nesnesi olarak döndürülür. Ancak bir x kompleks sayısının reel ve imajiner kısımları sırasıyla `x.real` ve `x.imag` komutlarıyla (Bir nesne üzerinde böyle çalıştırılan komutlara birer **metod** denir.) birer reel sayı (float) olarak elde edilebilir.

Terminal
```
>>> x.real
3.0
>>> x.imag
-2.0
```

Bunların dışında sonucun her zaman bir `complex` nesnesi olması bazı zorluklar yaratır. Örneğin bir kompleks sayı hesabının sonucunda imajiner kısım sıfır çıksa bile o sayı `float` nesnesi yerine hala bir `complex` nesnesi olarak döner.

Terminal
```
>>> z = x - 2*y
>>> z
(-1+0j)
>>> type(z)
<type 'complex'>
```

Hatta manuel olarak bile bu sayıyı bir `float` nesnesine dönüştüremeyiz. Bunu denersek Python yazılımı bize `complex` nesnelerinin `float` nesnelerine dönüştürülemeyeceğini anlatan bir hata mesajı gösterir.

Terminal
```
>>> a = float(z)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can't convert complex to float
```

Böyle bir sayıyı aşağıdaki gibi bir `float` veya `int` nesnesine dönüştürmek mümkündür.

Terminal
```
>>> a = z.real
>>> a
-1.0
>>> type(a)
<type 'float'>
```

Kompleks Fonksiyonlar

Python'da kompleks sayıları daha önce `math` modülünde gördüğümüz matematiksel fonksiyonlar içinde kullanamayız, o fonksiyonlar `float` nesneleriyle çalışacak şekilde tanımlanmışlardır. Mesela yukarıda tanımladığımız `x` kompleks sayısının sinüs değerini hesaplatmaya çalışırsak aşağıdaki gibi bir hata mesajı ile karşılaşırız.

Terminal
```
>>>import math
>>> math.sin(x)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can't convert complex to float
```

Kompleks değişkenler içeren matematiksel fonksiyonlar Python'un `cmath` (complex math) modülü içinde tanımlanmıştır ve kompleks matematiksel işlemler için bu modülü kullanmalıyız.

Terminal
```
>>> import cmath
>>> cmath.sin(x)
(0.5309210862485197+3.5905645899857794j)
```

Fakat `cmath` modülündeki fonksiyonlarla işlem yaparken de sonuçta imajiner kısım sıfır çıkınca bu bir `complex` nesnesi olarak döner. Bir hesaplama sonucunda imajiner kısım varsa bunun bir `complex`, yoksa da bunun bir `float` nesnesi olarak dönmesini arzularız. Bunun için `cmath` yerine `scipy` (SciPy, Scientific Python) paketini kullanmalıyız. Burada da tüm matematiksel fonksiyonlar vardır ve hepsi sonuçta imajiner kısım varsa `complex`, yoksa `float` nesnesi döndürecek şekilde tanımlanmıştır.

Terminal
```
>>> from scipy import sin, pi
>>> x = 3 - 2j
>>> sin(x)
(0.5309210862485197+3.5905645899857799j)
>>> sin(pi)
1.2246467991473532e-16
```

Yuvarlama Hataları

Bölümü bitirmeden önce biraz da yuvarlama hatalarından bahsetmek gerekiyor, aşağıdaki gözlemi yapalım.

Terminal
```
>>> a = 1.0/49
>>> b = 49
>>> a*b
0.9999999999999999
```

Buradaki hata programlama hatası değildir, bilgisayarların çalışma prensipleri ile ilgili temel bir durumdur bu ve tüm programlama dillerinde böyle durumlarla karşılaşılır. Bir reel sayı aslında çok sayıda basamaktan oluşur (rasyonel değilse sonsuz) ve bilgisayar hafızaları sınırlı

olduğundan sadece baştan bir kaç basamağı kullanılarak hafızada tutulurlar. Genellikle bilgisayarlarda 17 basamak kullanılır, dolayısıyla reel sayılarla (`float` tipi veri) hesaplama yaparken aslında bir miktar hata payı ile çalışıyor oluyoruz, bu tür hatalara **yuvarlama hatası** denir. `print` ifadesi ile bir float verisini yazdırırken kaç basamak gösterilmesi gerektiğini belirtmezsek, python az sayıda basamak kullanarak yazdırır ve yuvarlama hatalarını çoğu zaman farketmeyiz. Pythonda reel sayıları istenilen hata payı ile hesaplayabilen ve dolayısıyla sonuçta arzu edilen hata sınırları içinde kalmamızı sağlayan `decimal` modulü, ve `SciPy` paketi içinde bulunan `mpmath` modülü vardır. Fakat bunlara genelde fazla ihtiyaç duymayız çünkü nümerik yöntemlerin geliştirilmesinde yapılan hatalar yuvarlama hatalarından genellikle çok daha büyüktür.