

Pyhton Programlama - Seri II

Berkay Dedeoğlu

Nisan, 2016

İçindekiler

1	Giriş	2
2	Geçen Sayıda Neler Yaptık?	3
3	Python’da Şart İfadeleri	4
3.1	If Deyimi	4
3.2	Else Deyimi	6
3.3	Elif Deyimi	7
3.4	Şart İfadelerinde Diğer İşleçler	7
4	Uçbirimdekiler Uçar IDE’dekiler Kalır	9
4.1	IDLE	9
4.2	Ninja IDE	9
4.3	Geany	12
5	Yazılanların Kaydedilmesi ve Çalıştırılması	14
5.1	Uçbirim ile	14
5.2	Çift Tıklama ile	15
6	Programlamaya Devam	18
6.1	Len() Fonksiyonu	19
7	Python’da Tip Dönüşümleri	21
7.1	Sayı Değerleri (Integer)	21
7.2	Karakter Dizisi (String)	22
7.3	Ondalık Sayı (Float)	23
8	Son	25
9	Sorular	26

1 Giriş

Tekrar merhaba...

İnsanlar siyah ekranın üzerindeki yeşil - hızlı akan yazılardan, buna eşlik eden taramalı silah sesine benzeyen ve her tuştan farklı tonda ses çıkaran klavyeden; en önemlisi tüm bunları sağlayan bilgisayar başındaki o kişiden neden bu kadar korkar ki?

Geçmişte, geçmiş derken bilgisayarın henüz üniversite ve büyük kurumlara yayıldığı zamanlarda, bu durum çok normal karşılanıyordu.. Henüz grafik ekran yoktu, bu kadar iş yapan bilgisayar komutları yoktu.. Bu denli işe yarar program yazılacak diller bile yoktu...

İnsanlar neden bu denli korkar bilmiyorum ama bu korku bu durumu sağlayabilenler açısından çok büyük fırsattır. Karşısındaki insan henüz programlamanın bu kadar kolay olduğunu bilmiyor. Aslında yapılan iş basit... Bilgisayara onun anladığı dilden konuşacaksın bu kadar...

Programlama dili öğrenmek kolaydır. Asıl olay bu öğrenileni fikirlerinle birleştirebilmektir. Ancak bu şekilde bir hesap makinesi yapabilirsiniz.. Ve yine ancak bu şekilde bir füzeyi uzaya yollayabilirsiniz...

Lafı uzatmayalım, hemen öğrenmemize devam edelim.

2 Geçen Sayıda Neler Yaptık?

Geçen sayımızda programlamanın ne işe yaradığını, neden programlama yapıldığını, programların nasıl hazırlandığını, bilgisayarların bizi nasıl anlayacağını ve neden Python seçtiğimizi öğrendik.

Bununla birlikte Python'a ufak bi' giriş yapıp Python kodlarının sözdizimini anlayıp ileride bizleri neyin beklediğini anlamaya çalıştık.. Bu sayımızda Python'da daha da derinlere ineceğiz. Ve kesinlikle daha elle tutulur şeyler yapmayı deneyeceğiz.

3 Python’da Şart İfadeleri

Programlama yaparken şart ifadeleri programın gerçekten çok büyük bir bölümünü kapsar. Kullanıcının istekleri, işletim sisteminin uyumluluğu, donanımın uyumluluğu ve daha bir çok kontrol şart ifadeleri tarafından yapılır.

Bir robot tasarladığımızı düşünelim. Robotun yürüme fonksiyonlarını tanımlıyoruz. Robotun yürüme işini adam akıllı yapabilmesi için ona anlayacağı dilde (Makine dili ya da makine diline çevrilebilen bir dilde) şu ifadeleri söylememiz gerekir:

- Eğer karşında bir cisim varsa yönünü değiştir ve karşında bir cisim olup olmadığını kontrol et.
- Yok, karşında bir cisim yoksa yoluna devam et.

Bu şekilde robotumuz biryerlere çarpmadan ilerleyebilir. Farkındaysanız bunu şart ifadeleriyle sağladık. Yani robotun yürüme işini şarta bağladık. Ya da şöyle bir örnek verelim:

Bir program yazdık. Programın bir kısmında kullanıcının bastığı tuşa göre işlem yapacağız. ‘Ç’ tuşu programdan çıkmaya, ‘D’ tuşu devam etmeye yarasın. O halde programa yine anlayacağı dilde şöyle söylememiz gerekir:

- Eğer kullanıcı ‘Ç’ tuşuna basarsa programdan çık.
- Yok eğer kullanıcı ‘D’ tuşuna basarsa programa devam et.

Bu işlemi yapmak için yine şart ifadelerini kullanmalıyız.

Eğer şart ifadelerinin mantığı anlaşıldıysa şimdi bunu Python dilinde nasıl ifade edebileceğimizi görelim.

3.1 If Deyimi

Python’da şart ifadeleri bir bütündür. Genellikle tek kullanılmazlar. Ancak if deyimi şart ifadelerinin temelini oluşturur.

Şart ifadelerini ‘if’ deyimi ile başlatırız. If deyimini kullanmayı öğrenmeden önce girintilerde ilgili bir hatırlatma yapayım:

Python sözdizimi girintiler ile yazılır. Python’da ‘:’ ifadesinden sonra girinti gelmelidir. Eğer IPython3 kullanmıyorsanız ‘:’ ifadesinden sonra kendiniz girinti bırakmalısınız. Bunu 4 boşluk ya da ‘tab’ tuşu ile sağlayabilirsiniz. Daha detaylı bilgi için önceki sayıya bakabilirsiniz.

Bu ufak hatırlatmadan sonra if deyimini nasıl kullanacağımızı göstereyim:

```
1 if şartifadesi :
2     Çalışacak 1. kod
3     Çalışacak 2. kod
4
5     .
6     Çalışacak n. kod
```

Kodlarımızı bu formüle göre yazdığımızda eğer şart sağlanırsa altına yazdığımız kodlar çalışacaktır. Şimdi daha iyi anlamak için bir örnek verelim:

```
1 a = 5
2
3 if a == 5:
4     print('a değeri 5 olduğundan if ifadesinin içine girildi')
```

Bu kodları çalıştırdığımızda ‘a değeri 5 olduğundan if ifadesinin içine girildi’ çıktısını alacaksınız. Çünkü yazdığımız kodlar bizim dilimizde aşağıdaki ifadeye takabul ediyor:

- Eğer a değeri 5 ise ‘a değeri 5 olduğundan if ifadesinin içine girildi’ ifadesini döndür.

a değeri gerçektende 5 olduğu için if deyiminin altına yazdığımız kodlar çalıştı. Birde şöyle deneyelim:

```
1 a = 7
2
3 if a == 3:
4     print("a değeri 3'tür. Bu yüzden bu yazıyı görüyorsunuz.")
```

Bu kez hiçbir çıktı almadık. Çünkü a'nın değeri 3 değil. Yani 'if' deyimi karşısındaki ifade doğru olduğunda bünyesindeki kodların çalışmasına izin veriyor.

Bu arada if deyimini kullanırken eşitlik belirtmek için '==' ifadesini kullanıyoruz. Python'da bu tip ifadelere işleç denir. İşleçleri ileride detaylı biçimde göreceğ olsakta şimdilik sadece göstermem şart ifadelerini anlamak için gereklidir.

- '=' : Eşitlik bildirir. İki ifadenin birbirine eşit olduğunu kontrol eder.
- '!=' : Eşitsizlik bildirir. İki ifadenin birbirine eşit olmadığını kontrol eder.
- '<' : Küçüklük bildirir. İlk ifadenin ikinci ifadeden küçük olduğunu kontrol eder.
- '>' : Büyüklük bildirir. İlk ifadenin ikinci ifadeden büyük olduğunu kontrol eder.
- '<=' : Küçük - eşitlik bildirir. İlk ifadenin ikinci ifadeden küçük olduğunu ya da eşit olduğunu kontrol eder.
- '>=' : Büyük - eşitlik bildirir. İlk ifadenin ikinci ifadeden büyük olduğunu ya da eşit olduğunu kontrol eder.

Aslında kontrolü bu işleçler yapar. Yani eğer bu işleçler yaptıkları kontrole göre True ya da False değeri döndürürler. Bunu, aşağıda da görebilirsiniz:

```
1 >>> a = 3
2 >>> b = 4
3 >>> c = 'qwe'
4 >>> d = 'qwerty'
5 >>> e = 'qwe'
6 >>> f = 3
7
8 >>> a == f
9 True
10 >>> b < a
11 False
12 >>> c == e
13 True
14 >>> c <= e
15 True
16 >>> d > e
17 True
18 >>> c <= d
19 True
20 >>> c == d
21 False
```

Eğer bu işleçlerin yaptığı kontrollerden olumlu sonuç alınıyorsa True değeri döndürülür. Eğer 'if' deyiminin karşısında True değeri varsa aşağıdaki kodlar çalıştırılır. Yani 'if' deyimini şu şekilde de kullanabilirsiniz:

```
1 if True:
2     print('If deyiminin karşısındaki ifade doğru')
```

Bu şekilde if deyimi kontrol yapmadan bünyesindeki kodları çalıştırır. Python'da True ifadesi sayısal biçimde 1 olarak gösterilir. Yani bir önceki örnekle aşağıdaki örnek tamamen aynıdır:

```
1 if 1:
2     print('İf deyiminin karşısındaki ifade doğru')
```

Dilerseniz 'if' deyiminin karşısına bu kez de False değeri verelim. Aşağıdaki iki örnekte tıpkı yukarıdakiler gibi aynı şeyi ifade eder:

```
1 if 0:
2     print('İf deyiminin karşısındaki ifade doğru')
```

```
1 if False:
2     print('İf deyiminin karşısındaki ifade doğru')
```

Yukarıdaki kodlardan birini denediyseniz hiçbir çıktı alınmadığını görmüşsünüzdür.

Şimdi çok küçük bir parola denetim programı yapıp konumuza devam edelim:

```
1 şifre = 'abcde555'
2
3 giriş = input('Lütfen şifreyi girin: ')
4 #Bu kısımda istediğiniz bir şifre deneyin...
5
6 if şifre == giriş:
7     print("Giriş Başarılı")
```

Eğer input fonksiyonuna verdiğiniz cevap başta tanımladığınız şifrenin aynısı ise 'Giriş Başarılı' ifadesi dönmeli.

İf deyimi hakkında oldukça şey öğrendik. Şimdi şart ifadelerinin diğer deyimine geçelim.

3.2 Else Deyimi

'if' kelimesi İngilizce de 'eğer' anlamına gelirken, 'else' ifadesi 'aksi halde' anlamına gelir. Python dilinde de bu deyimleri gerçek anlamlarında kullanırız.

'Else' deyimine şöyle bir örnek vereyim:

- a değeri 7 'dir.
- eğer a değeri 3 ise ekrana 'Doğru' yazdır. Aksi halde ekrana 'Yanlış Cevap' yazdır.

Şimdi bu ifadeyi Python'da yazalım:

```
1 a = 7
2
3 if a == 3:
4     print('Doğru')
5 else:
6     print('Yanlış Cevap')
```

Yukarıdaki kodlardaki anlaşıldığı gibi eğer şartımız doğru değilse hiçbir kontrol yapmadan 'else' deyiminin altındaki kodlar çalıştırılıyor. Yani a değeri 4,6,88,100 ve 3 haricindeki hangi sayı/değer olursa olsun ekranda 'Yanlış Cevap' yazısı döndürülecek.

Kısacası if deyiminin kodları çalıştırılmazsa else deyiminin kodları çalıştırılır. If deyiminin kodları çalıştırılırsa else deyiminin kodları çalıştırılmaz. Ufak bir örnek verip devam edelim:

```

1 şifre = 'SuDoPyThoN55'
2
3 alınan = '123456'
4
5 if şifre == alınan:
6     print('Tebrikler sisteme başarı ile giriş yaptınız...')
7 else:
8     print('Malesef hatalı giriş yaptınız.')
```

3.3 Elif Deyimi

Elif deyimi else ve if deyiminin karışımıdır. Genellikle if ve else deyimlerinin arasına girer. If deyiminin karşısındaki şartın yanlış olduğu durumda işletilir. Else deyiminden farkı ise karşısına şart almasıdır.

Biraz havada kaldı gibi :) Hemen bi' örnek vereyim:

```

1
2 print('2+7' + 'ya da ' + '5 + 5' + '7 + 5' + ' sorularından birini cevaplayın')
3
4 cevap = input('Cevap: ')
5 #Cevap bu fonksiyona verilir.
6
7 if cevap == '9':
8     print('1. soruya verdiğiniz cevap doğru')
9 elif cevap == '10':
10    print('2. soruya verdiğiniz cevap doğru')
11 elif cevap == '12':
12    print('3. soruya verdiğiniz cevap doğru')
13 else:
14    print('Hiçbir Soruya doğru cevap veremediniz.')
```

Bu örnek if, elif ve else deyimlerini anlatıyor. Elif deyiminin sınırı yoktur istediğiniz kadar elif kullanabilirsiniz. Ancak iyi bir programcı bunu en aza indirmelidir.

Ben elif yazmam onun yerine hepsine if yazarım diyorsanız size önerim yukarıdaki örneğin aynısını sadece elif deyimlerini if olarak değiştirip denemenizdir.

Bir sorun var değil mi? Eğer hala çözemediyseniz foruma bekleriz .

3.4 Şart İfadelerinde Diğer İşleçler

Şart ifadelerinde kullandığımız ve gerçekten işe yarayan iki önemli işlecimiz daha var. Kullanımı oldukça basit; 'and' ve 'or'.

- **'and':** 've' anlamına gelir. Sağında ve solundaki her iki ifadenin doğru olması gerekir. Aksi takdirde True değeri dönmez.
- **'or':** 'veya' anlamına gelir. Bağladığı ifadelerden en az birinin doğru olması True değerinin dönmesi için yeterlidir.

Aşağıdaki örneklere bakın:

```

1 a = 3
2 b = 5
3 d = 'a'
4 e = 'li'
5 f = 8
6
```



```
7 | #-----True Değerleri-----#  
8 | (a + b == 8) and (b + a == 8)  
9 | (f - a == 3) or (d + e == 'ali')  
10 | ( (8 / 4 == 4) or (a + 4 == 7) ) and (d + e == 'ali')  
11 |  
12 | #-----False Değerleri-----#  
13 | (e + d == 'ali') and (b + a == 8)  
14 | (f - a == b-1) or (e + d == 'ali')
```

Bu işlemler yaptığı kontroller sonucunda 'True' ya da 'False' değeri döndürüyor. O halde biz bu işlemleri şart ifadelerinde rahatlıkla kullanabiliriz.

4 Uçbirimdekiler Uçar IDE'dekiler Kalır

Bu kısımla birlikte programcılık serüvenimizde adeta bir çağı kapatıp yeni bir çağ açacağız. Neden bu başlığı bu kadar büyüttüm? Çünkü bu zamana kadar Python'da yaptığımız her şey başlıkta yazdığı gibi uçbirimdeydi (IPython'da python uçbirimi olarak görülebilir) ve uçtu. :) Ayrıca açık konuşayım yaptıklarımız çokta işe yaramıyordu. Aslında bu bildiklerimizle çok güzel uygulamalar yapabiliriz ama yorum satırında çalışmak bize büyük engel.

Esasında bu kısımdan önceki örnekler bu engelden ötürü hep aynı şeylerin etrafında dönüyordu. Ama artık çok daha geniş kapsamlı ve çok daha kaliteli uygulamalar yapabileceğiz.

Bu bahsettiklerimin tamamının sonu bir IDE'ye varıyor. Peki IDE nedir? Türkçe'ye tümleşik geliştirme ortamı olarak çevrilen ve kodlamacılara kod yazımı süresince yardımcı olan metin editörleridir. Aslında kimi geliştiriciler programlamaya yeni başlayanların çok gelişmiş bir IDE kullanmamasını önerirler. Çünkü çok gelişmiş IDE'ler ciddi anlamda leblebiyi anlayacak düzeyde olabilirler. Bu durumun öğrenmeyi zorlaştırdığı söylenir. Yani küçük bir çocuğun ödevlerine yardım edeyim ayağına ödevlerin tamamını tek başına bitirmek gibi. Bu geliştiriciler çok gelişmiş IDE'leri birde çok profesyonelce tasarlandığı için tavsiye etmezler. Çok gelişmiş IDE'ler büyük projeler için geliştirildiğinden küçük denemelerde fazla sayıda klasör, bilinmeyen terimler, bilinmeyen dosya uzantıları gibi fazla detayları küçük bir deneme programı için oluşturur. Bu da tedirginlik yaratabilir.

Öte yandan bir grup geliştirici ise çok gelişmiş IDE'lerin yeni başlayan programcıların öğreniminde bir sakınca olmadığını hatta yardımcı olduğunu savunur. Akıllı kod renklendirme, düzenli görünüm, her şeyin elin altında olması gibi özellikler savunmalarını oluşturur. Aslında bu da doğru bir yaklaşımdır.

Bana kalırsa her iki görüşte doğrudur. Bu yüzden bu güne kadar kullandığım ve memnun kaldığım tüm IDE'leri anlatacağım. İsteyen kişi istediği IDE'yi seçebilir. Bana kalırsa bu seviye için ilk anlatacağım IDE olan IDLE bizler için en uygundur. Dedğim gibi tercih sizin...

4.1 IDLE

IDLE Python'un resmi olarak desteklediği çok gelişmiş olmayan her seviyedeki geliştirici için kullanılabilir ve Python'a tam olarak uyumlu bir tümleşik geliştirici ortamdır.

Kullanımı oldukça basittir. İndirdiğinizde Python'a tam uyumlu şekilde gelir. İçinde birde gelişmiş bir Python yorumlayıcı bulundurulur. Programı ilk açtığınızda sizi bu yorumlayıcı karşılar. CTRL-N tuş kombinasyonu ile kodların yazılacağı kısma ulaşabilirsiniz. Yazdığınız programı F5 tuşuna basarak kaydedip çalıştırabilirsiniz.

Ubuntu ve türevlerinde bu uygulamayı kurmak için:

```
1 sudo apt-get install idle-python3.4
```

4.2 Ninja IDE

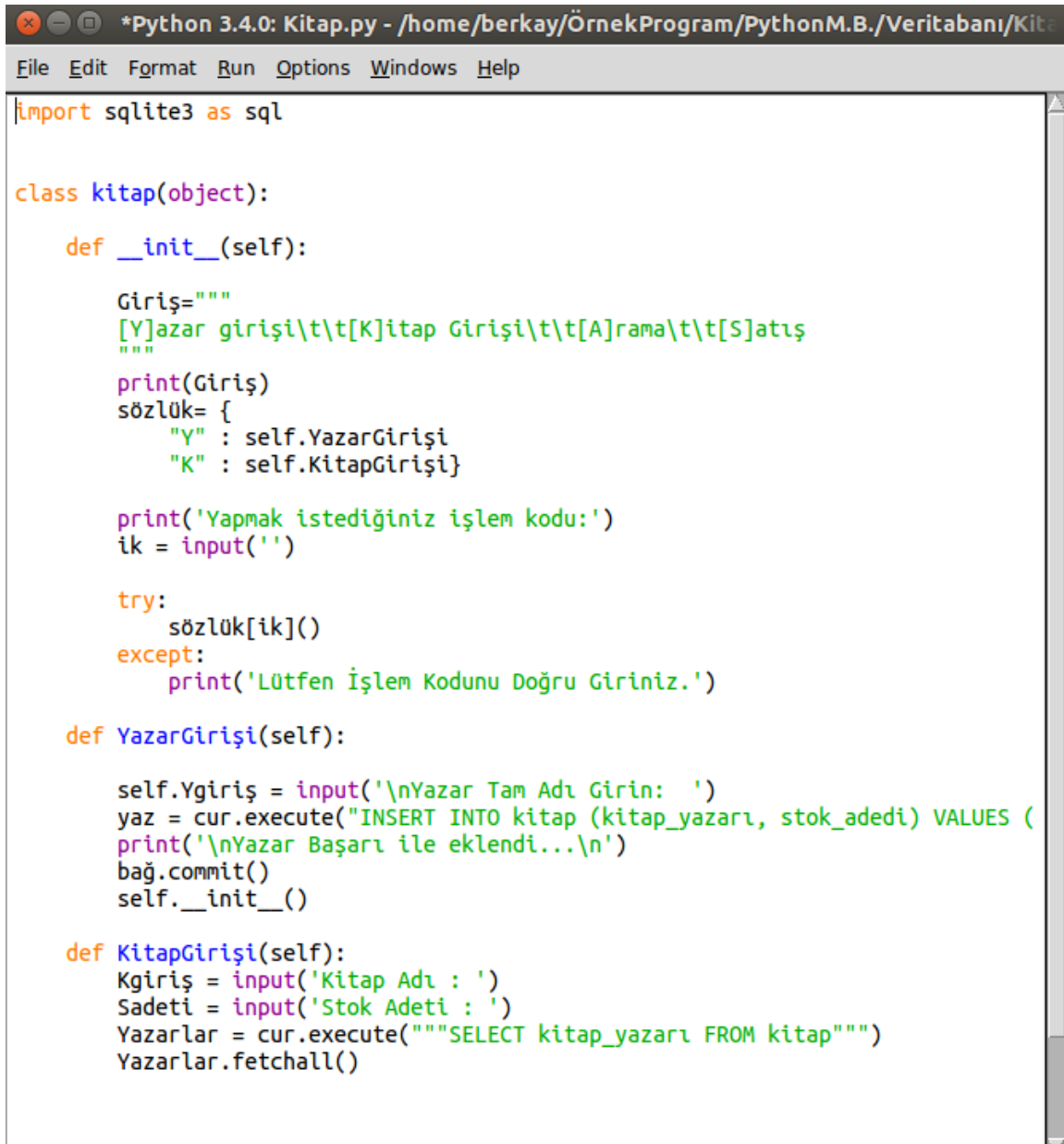
Ninja IDE açık kaynak kodludur. Python için düzenlenmiştir. IDLE gibi bu yazılım da Python ile yazılmıştır. Gelişmiş bir IDE'dir. Python projeleri için alışıldığında vazgeçilmeyecek bir programdır.

Kurulduğunda sizi yukarıdaki gibi bir başlangıç ekranı karşılar. İyi seviyedeki programcılar için oldukça yararlıdır. Eklenti desteği vardır. Yazılan betikler F8 tuşu ile çalıştırılabilir.

Kaliteli bir kod renklendirmesi vardır. Hızlı ve kullanıcı dostudur. Kurulduktan sonra hemen program yazmaya başlayabilirsiniz. Herhangi bir özelleştirme gerektirmez.

İndirme komutu:

```
1 sudo apt-get install ninja-ide
```



```
*Python 3.4.0: Kitap.py - /home/berkay/ÖrnekProgram/PythonM.B./Veritabanı/Kitap.py
File Edit Format Run Options Windows Help

import sqlite3 as sql

class kitap(object):

    def __init__(self):

        Giriş="""
        [Y]azar girişi\t\t[K]itap Girişi\t\t[A]rama\t\t[S]atış
        """
        print(Giriş)
        sözlük= {
            "Y" : self.YazarGirişi
            "K" : self.KitapGirişi}

        print('Yapmak istediğiniz işlem kodu:')
        ik = input('')

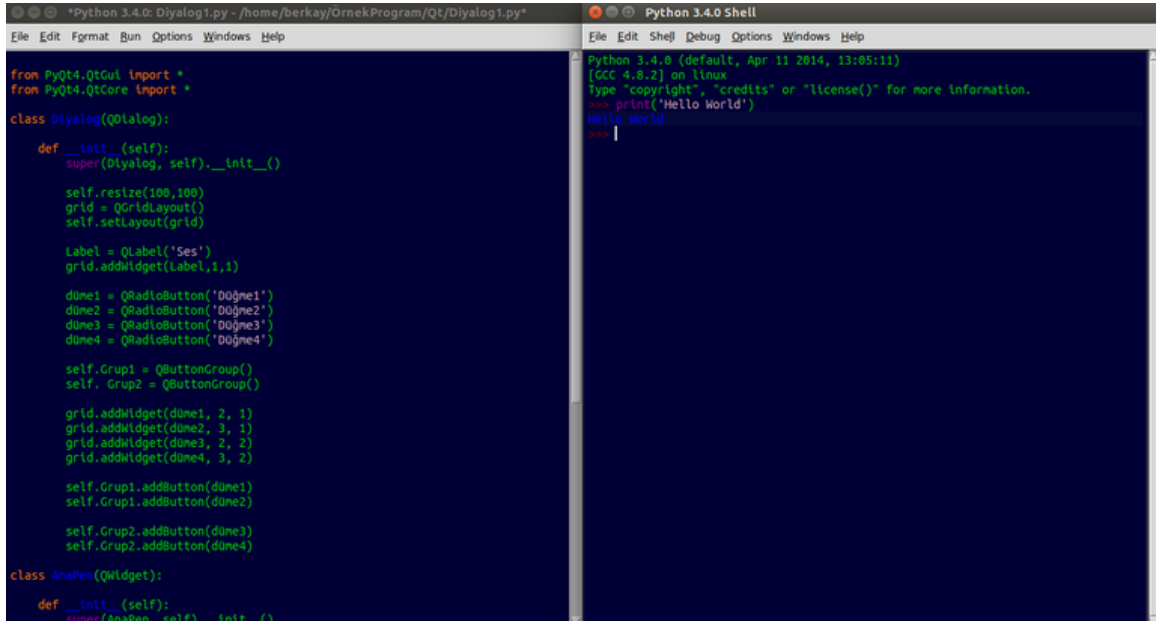
        try:
            sözlük[ik]()
        except:
            print('Lütfen İşlem Kodunu Doğru Giriniz.')

    def YazarGirişi(self):

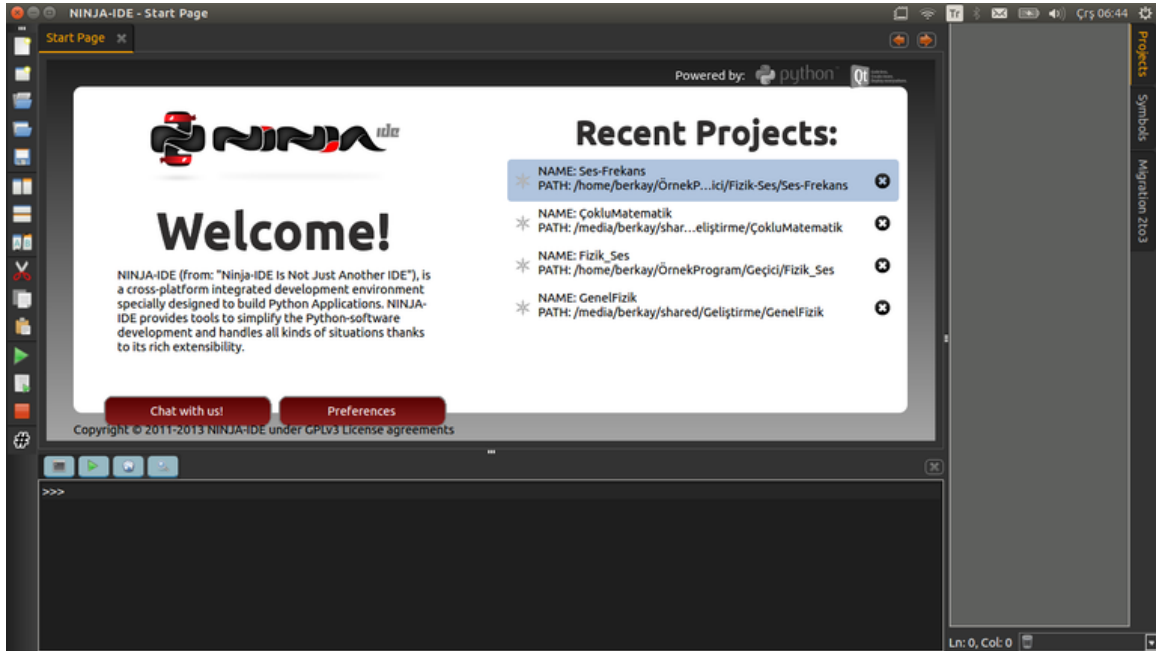
        self.Ygiriş = input('\nYazar Tam Adı Girin: ')
        yaz = cur.execute("INSERT INTO kitap (kitap_yazarı, stok_adedi) VALUES (
        print('\nYazar Başarı ile eklendi...\n')
        bağ.commit()
        self.__init__()

    def KitapGirişi(self):
        Kgiriş = input('Kitap Adı : ')
        Sadeti = input('Stok Adeti : ')
        Yazarlar = cur.execute("""SELECT kitap_yazarı FROM kitap""")
        Yazarlar.fetchall()
```

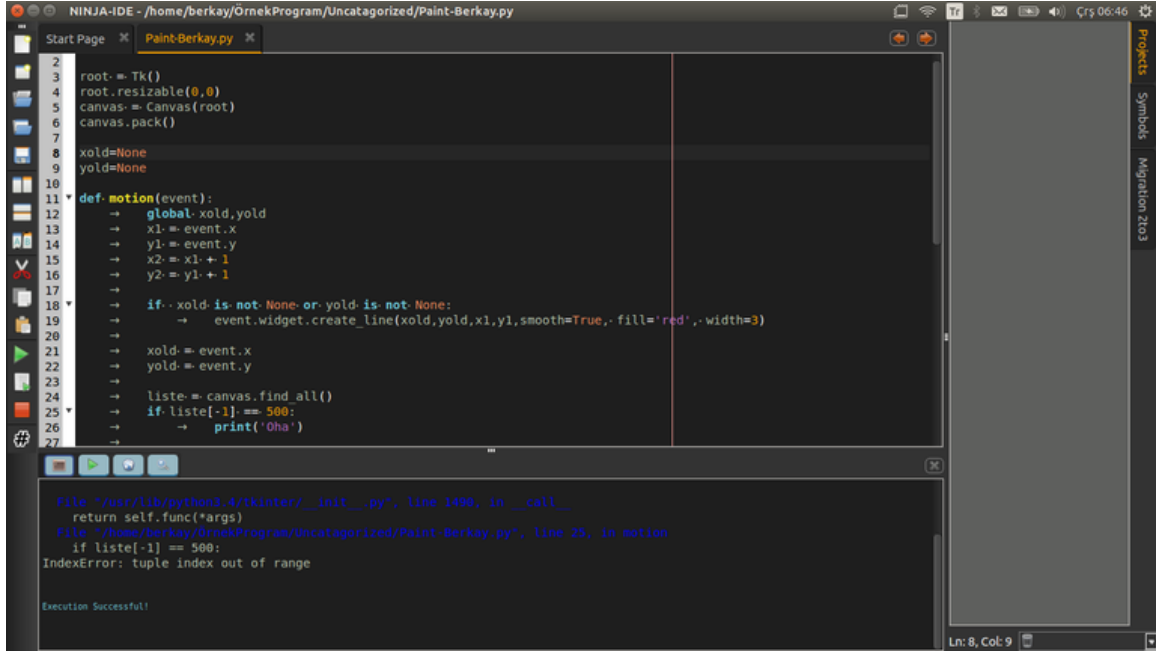
Şekil 1:



Şekil 2:



Şekil 3:



Şekil 4:

4.3 Geany

Geany oldukça hafif bir IDE'dir. Program sadece Python için yazılmasa da Python için rahatlıkla ve zorlanmadan kullanılabilir. Acemi kullanıcıların kafasını karıştırmayacak düzeydedir.

Eğer yukarıda bahsettiğimiz meseleye göre gelişmiş IDE kullanmak istemiyorum ama kendimi aşırı biçimde de yormak istemiyorum dersiniz ve IDLE'yi de bağenmediyseniz Geany ikinci alternatifiniz olabilir. F5 tuşu ile yazdığınız betik çalıştırılır.

İndirip kurmak için aşağıdaki komutu çalıştırmalısınız:

```
1 sudo apt-get install geany
```

Diğer bazı IDE'leri ve Python'da kullanımlarını forumda Python bölümünde bulabilirsiniz.

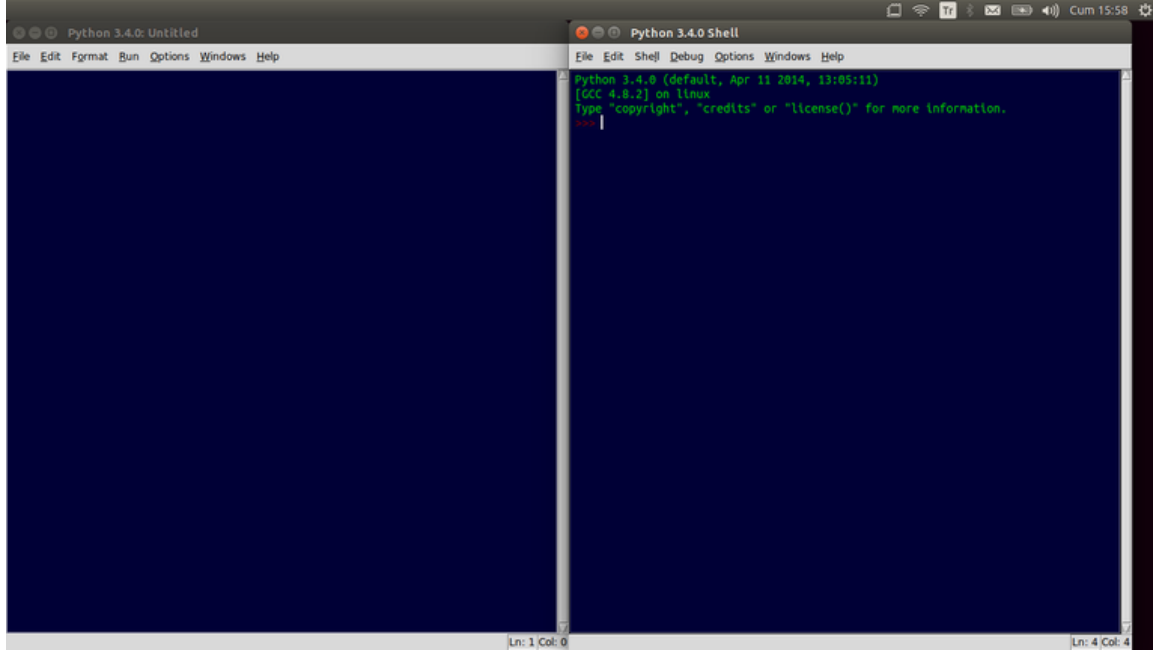
```
deneme.pyw - /home/berkay/ÖrnekProgram/Geçici - Geany
1 from PyQt4.QtGui import *
2 from PyQt4.QtCore import *
3
4 class Dialog(QDialog):
5
6     def __init__(self, ebe=None):
7         super(Dialog, self).__init__(ebe)
8
9         self.ebeveyn = ebe
10        self.setAttribute(Qt.WA_DeleteOnClose)
11
12        Layout = QGridLayout()
13
14        Layout.addWidget(QLabel('<center><b>Yazı Tipi:</center></b>'), 1,1)
15        Layout.addWidget(QLabel('<center><b>Yazı Rengi:</center></b>'), 1,1)
16
17        self.YazıTipi = QFontComboBox()
18        Layout.addWidget(self.YazıTipi, 1, 2)
19        self.YazıTipi.setCurrentFont(QFont(self.ebeveyn.tlp))
20
21        self.YazıRengi = QColorComboBox()
22        self.YazıRengi.addItem(['Kırmızı', 'Sarı', 'Mavi', 'Mor', 'Yeşil', 'Siyah', 'Turuncu'])
23        Layout.addWidget(self.YazıRengi, 2,2)
24
25        self.Sözlük = {
26            "Kırmızı": "red", "Sarı": "yellow",
27            "Mavi": "blue", "Mor": "purple",
28            "Yeşil": "green", "Siyah": "black",
29            "Turuncu": "orange"
30        }
31
32        kutu = QHBoxLayout()
33        Layout.addLayout(kutu,3,1,1,2)
34
35        Uygula = QPushButton('Uygula')
36        kutu.addWidget(Uygula)
37
38        Vazgeç = QPushButton('Vazgeç')
39        kutu.addWidget(Vazgeç)
40
41        self.setLayout(Layout)
42
43        self.connect(Uygula, SIGNAL('pressed()'), self.devam)
```

satır: 1/98 kol: 0 seç: 0 INS TAB mode: Unix (LF) kodlama: UTF-8 dosyatürü: Python alan: bilinmeyen

Şekil 5:

5 Yazılanların Kaydedilmesi ve Çalıştırılması

Yukarıda bahsettiğim programlar aracılığıyla programlarımızı yazabiliriz. Biz bir süre boyunca anlatırken IDLE programını kullanacağız. Şimdi IDLE programını açalım.



Şekil 6:

Karşımıza yukarıdaki gibi bir ekran çıkmış olmalı. Kodlarımızı buraya yazmayalım. Çünkü burası Python kabuğu, yani Python yorumlayıcısı. CTRL-N tuşları ya da File>>New File yolunu izleyerek kodları yazacağımız alana geçelim.

Evet artık programlarımızı oluşturacağımız ekrana ulaştık. Bundan sonra kodlarımızı buradan yazacağız. Her seferinde CTRL-N ya da File>>New File yapmak istemiyorsanız; bu ekranda Options>>Configure IDLE>>General yolunu izleyip, bu ekranda ‘Startup Preferences’ bölümünde ‘Open Edit Window’ kutucuğunu işaretleyebilirsiniz.

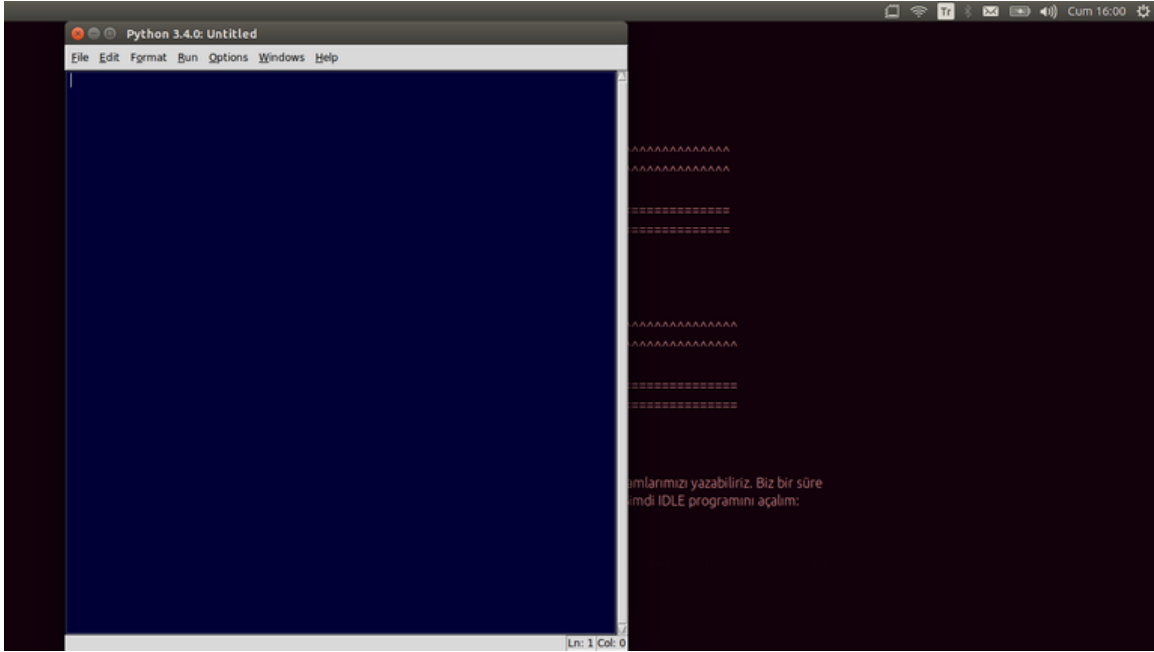
Şimdi basit bir program yazalım bu ekrana:

```
1 yazdır = """
2 Merhaba Dünya
3 Ben bir Python3 programıyım.
4 """
5
6 print(yazdır)
```

Programı açıklamaya gerek yok. Tabi bir istisna hariç; eğer bir önceki sayıda bahsettiğimiz **string** ifadelerini üç tırnak ile oluşturuyorsak dilediğimiz gibi alt satıra geçebiliriz, boşluk bırakabiliriz. Bu durum sadece **string** ifadesini üç tırnak ile oluşturduğunuzda geçerlidir. Şimdi bu programı çalıştıralım:

5.1 Uçbirim ile

Python programlarını uçbirim ile çalıştırmak en makul yol denilebilir. Çünkü eğer hata varsa uçbirim ekranından hatayı rahatlıkla görebiliriz.



Şekil 7:

Yazdığımız programı uçbirimden çalıştırmak için öncelikle bir yere kaydetmeliyiz. Bunun için IDLE ekranında CTRL-S tuş kombinasyonunu ya da File>>Save File yolunu uygulamalıyız. Karşımıza çıkan diyalog pencerede dosyanın kaydedileceği yer belirlenir ve uygun bir isimle dosya kaydedilir.

Programı “/home/Kul...Adı/Programlarım/” şeklinde bir dizine ‘deneme.py’ adıyla kaydettiğinizi varsayarsak (Tabi burada ‘Kul...Adı’ sizin kullanıcı adınızdır) program şu komutlarla çalıştırılır:

```
1 cd ./Programlarım/  
2 python3 deneme.py
```

Bu komutları açıklayacak olursak;

Öncelikle uçbirimden dosyanın olduğu yere gidilir. Bu cd komutuna verilen bir parametre ile yapılır. ‘./’ ise o anki dizine kadar olan yerleri yazmamak içindir. Yani ‘./’ yazdığımızda “/home/Kul..Adı” yazmama gerek kalmaz.

Daha sonra programımızı python3 komutuna belirterek çalıştırıyoruz.

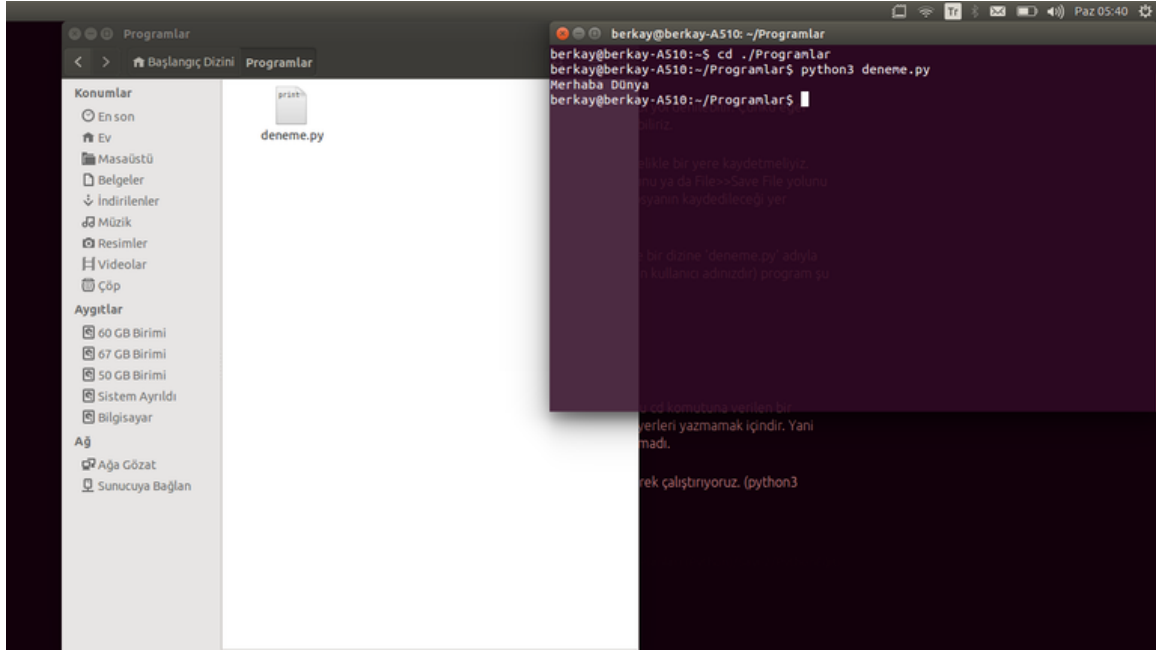
```
1 python3 deneme.py
```

Yukarıdaki ekranı görüyorsanız programı başarı ile çalıştırdınız demektir. Programımız burada çok basit bir program olduğu için uçbirim ekranında sadece ‘Merhaba Dünya’ yazdırıp kendini sonlandırıyor.

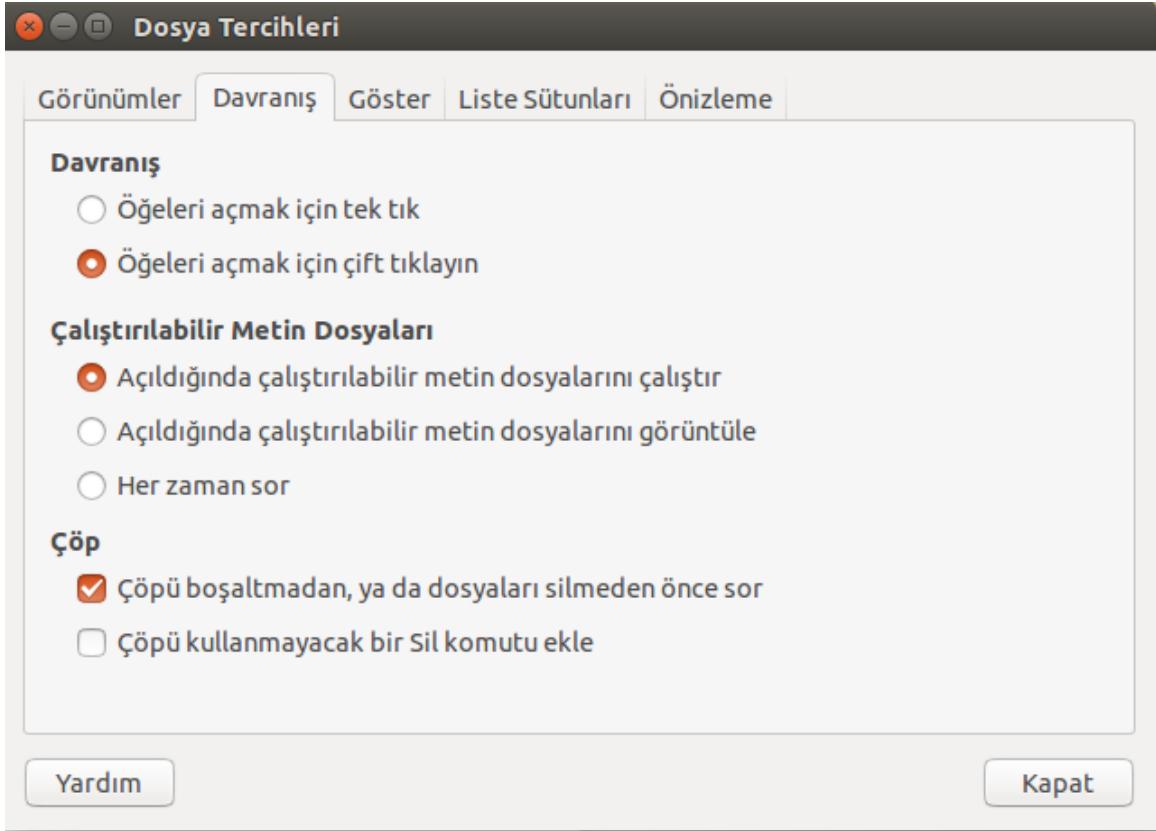
5.2 Çift Tıklama ile

Linux dağıtımlarında bizim hazırladığımız tarzdaki metin belgelerini çalıştırabilmek için bu dosyayı çalıştırılabilir yapmamız yeterlidir. Ancak Ubuntu’nun da varsayılan olarak kullandığı Nautilus dosya yöneticisi, bu tip dosyaları, çalıştırılabilir olmasına rağmen program gibi çalışmasına izin vermez. Eğer Nautilus dosya yöneticisini kullanıyorsanız ve programlarınızı çift tıklama ile çalıştırmak istiyorsanız Nautilus’un bu özelliğini devre dışı bırakmalısınız.

Eğer Nautilus kullanıyorsanız, yukarıdaki özelliği devre dışı bırakmak için Nautilus dosya yöneticisini açıp *Değiştir>>Tercihler>>Davranış* yolunu izlemeniz gerekir. Karşınızdaki pencere aşağıdaki gibi olmalıdır:



Şekil 8:



Şekil 9:

Bu pencerede **Çalıştırılabilir Metin Dosyaları** kısmında üç seçenek bulunuyor. Bunlar arasından son seçenek olan *'Her zaman sor'u* seçip kapat düğmesine basın.

Bu işlemi yalnızca bir kez yapmanız yeterlidir. Daha sonra programımıza çalışma yetkisi vermeliyiz. Biraz önceki programdan devam edelim. Programımız *Programlarım* klasöründeydi ve adı *deneme.py* idi. Şimdi bu programı çalıştırılabilir yapalım:

İki yol var. Dilerseniz .py uzantılı dosyaya sağ tıklayıp özellikler diyin. *Erişim Hakları* sekmesinden *Dosyanın bir program gibi çalışmasına izin ver* kutucuğunu işaretleyip kapatın. Ya da uçbirimde Programlarım dizinine ulaşp şu komutu verin:

```
1 chmod +x deneme.py
```

Bu şekilde dosyamızı çalıştırılabilirde yaptık. Bu sayede dosyaya çift tıklayınca bilgisayar bu dosyayı çalıştırmaya çalışacak.

Ancak denediyseniz görmüşsünüzdür ki program çalışmadı. Çünkü işletim sistemine bu programı nasıl çalıştıracağına dair bir ipucu vermiyoruz. O halde programın kodlarını tekrar açıp ipucumuzu programın en başına yazalım:

```
1 #! /usr/bin/env python3
```

Bu yorum satırına benzesede yorum satırı değildir. İşletim sistemi için önemli bir satırdır. Programı python3 ile çalışacağını bildirir. Tüm bu işlemlerden sonra program çift tıklanınca ve onay kutusundan **Uçbirim ile Çalıştır** düğmesine basınca program çalışır.

Tabi bizim yazmış olduğumuz programda bir istisnai durum var. Programlar işlerini bitirene kadar çalışırlar. İşlerini bitirir bitirmez kapanırlar. Ee bizim programımızın işi uçbirime 'Merhaba Dünya' yazdırmak. Bu işini bitirince kapanıyor. Bizde ne olup bittiğini göremiyoruz. Bu yüzden programın sonuna birde şu kodu ekleyelim:

```
1 input()
```

Artık programımızı çift tıklayarak çalıştırabiliriz. Program, biz bir tuşa basana kadar kapanmaz.

6 Programlamaya Devam

Programlarımızı kaydetmeyi öğrenmişken hemen bir örnek yapalım ve bu işe ısınalım.

```
1
2 #!/usr/bin/env python3
3
4 """
5 Program ilk program olduğu için geçmişte öğrendiklerimizi pekiştirme
6 amaçlıdır. Bir amaca hizmet etmeyen bu program kullanıcıdan kod olarak
7 aldığı veriyi ekrana yazdırır.
8 """
9
10
11 print("""
12 —Programa Hoşgeldiniz—
13
14 Lütfen ekrana yazdırmak istediğiniz şeyin kodunu girin.
15 Eğer eğer kendiniz bir şeyler yazdırmak istiyorsanız '6' kodunu
16 girebilirsiniz.
17
18 1 → 'Merhaba Dünya'
19 2 → 'Hayat Nasıl'
20 3 → 'Ben Python dili ile yazıldım'
21 4 → 'İyi Bayramlar'
22 5 → 'İyi geceler'
23 6 → Kendim Gireceğim
24 """)
25
26 #Programımızın giriş yazısı yazıldı.
27
28 giriş = input('Kod: ')
29
30 if giriş == "1":
31     print('Merhaba Dünya')
32
33 elif giriş == "2":
34     print('Hayat Nasıl')
35
36 elif giriş == "3":
37     print('Ben Python dili ile yazıldım')
38
39 elif giriş == "4":
40     print('İyi Bayramlar')
41
42 elif giriş == "5":
43     print('İyi Geceler')
44
45 elif giriş == "6":
46     giriş2 = input('Gireceğiniz Cümle: ')
47     print(giriş2)
48
49 else:
50     print("Yanlış bir giriş yaptınız.")
51
52 input('Çıkış yapmak için bir tuşa basın...')
```

Şimdi bu programı detaylıca açıklayalım:

Programımızın ilk satırı malum; işletim sisteminin programı nasıl çalıştıracağını bildiriyor. Yani eğer çift tıklarak çalıştırmayacaksanız bunu yazmanız zorunluluk değil. Bu satırın altında `"""` işareti ile başlattığımız bir string tipi var. Aslında biz bu ifadeyi yorum satırı olarak kullandık. Programımızın çalışma düzenini, amacını, başlangıç tarihini, bitiş tarihini v.s. burada belirtebiliriz. Zorunluluk değildir. Ancak kodları tekrar okuduğunuzda ya da başka biri okuduğunda iyi bir rehber olacaktır. Bu kısmı yorum satırı işareti olan `#` ile de elirteildik

ancak bu kez her yeni satırda yeni bir yorum satırı işareti (#) bırakmak zorunda kaldık. Her ikisine de sıklıkla rastlayacaksınız.

Bir altında bildiğimiz bir fonksiyon olan *print* var. İçindeki yazı, programımız başladığında ekranda gösterilecek yazıdır. Yine düzenli gözükmeleri için (Alt satıra rahat geçebilmek için) `"""` işareti kullandım. Bir altta bir yorum satırı... Bunu açıklamama gerek yoktur umarım :)

Hemen altında bir *input* fonksiyonu... Bu fonksiyon, daha alt satırlarda kullanmak için kullanıcının gireceği kodu istiyor. Yani bu kod parçacığı sayesinde yorumlayıcı önce bir üstteki giriş yazısını andından **Kod:** soorgusunu gösterecek. Kullanıcı bu sorguya cevap vermelidir.

Cevap verip 'Enter' tuşuna bastığı anda bir alttaki kod çalışır. Bu bir şart ifadesidir. Ve bunu dilimize 'Eğer kullanıcının girdiği değer 1 ise' diye çevirebiliriz. Burada '1' sayısını tırnak içine almamın sebebi input fonksiyonundan aldığımız cevap bize string şeklinde ulaştırılır. Yani eğer bu şart ifadesinde 1 sayısını tırnak içine almasaydım hata alırdım.

Eğer kullanıcı '1' kodunu seçerse ekrana 'Merhaba Dünya' yazılması için bir print fonksiyonu yazdım.

Diğer şart ifadelerini yazmamdaki amaç ilk ifadeyi yazmamdaki amaç ile aynı. Ancak diğerlerinde 'elif' deyimini kullandığıma dikkat edin.

Biraz sonra 6. şart deyimini görüyoruz. Bu diğerlerinden farklı ama yine çok basit. Bu kez kullanıcının '6' kodunu seçmesi durumunda kullanıcıya bir soru daha soruyorum (input ile). Kullanıcıdan aldığım cevabı giriş2 olarak isimlendiriyorum. Bu şekilde kullanıcının girdiği yazı dizisini bir alt satırda ekrana yazdırıyorum.

Birde else deyimi var. Burada kullanıcının yanlış bir şey yaptığını ona bildirmeye çalışıyorum. Bu satırı yazmamış olsaydım ve kullanıcı kod olarak '50' sayısını verseydi program hata verip kapatırdı. Yani else deyimi ile 1,2,3,4,5,6 haricindeki tüm kodları kapsayan bir elif deyimi yazmış gibi oluyorum. Kullanıcı bu bu sayılar dışında bir şey girdiğinde ona bunun yanlış olduğunu bildiriyorum.

Ve son olarak bir input fonksiyonu görüyorsunuz. Bu fonksiyonun mantığı programın çat diye kapanmasını önlemek. Örneğin kullanıcı 1 kodunu seçti bu durumda hemen birinci şart ifadesi çalışıyor. Diğer şart ifadeleri uygun olmadığından geçiyor. Program tam kapanacakken input fonksiyonu onu engelliyor. Bu engel olmazsa kullanıcı daha 'Merhaba Dünya' yazısını göremeden program hızla kapanacak.

Ve bu şekilde program sonlanıyor. Şimdi bu bilgilerle programı tekrar çalıştırın. Eminim her şey yerine oturmuştur. Eğer bu konuda yanılıyorsanız sorularınızı forumun Python bölümünde sorabilirsiniz. :)

6.1 Len() Fonksiyonu

Programlamaya bir fonksiyon ile devam edeceğiz. Bu fonksiyon kullanımı açısından daha önce detaylıca öğrendiğimiz 'print' ve 'input' fonksiyonlarına benzer.

Bu fonksiyonun amacı, ona verdiğimiz değerın kaç karakterden oluştuğunu bize belirtmektir. Şimdi bir örnek verelim. Bu örneği herhangi bir IDE ile kaydetmenize gerek yok IPython3 ile çalıştırabilirsiniz:

```
1 >>> Değer = "Bu ifade 26 karakterlidir."
2 >>> len(Değer)
3 26
```

Gördüğünüz gibi önce bir string ifadesi oluşturduk. Daha sonra bu string ifadesini bir nesneye atayıp 'len' fonksiyonuna verdik. 'len' fonksiyonu bu nesneyi işleyerek 26 diye bir sayı değeri döndürdü. Eğer boşlukları da dahil ederek saymışsanız gerçektende bu ifade 26 karakterden oluşuyor. Bu ifadeyi nesneye tanımlamak zorunda değildik. Yani şu biçimde de len fonksiyonu işini başarı ile yapar:

```
1 >>>len("Bu ifade 26 karakterlidir.")
2 26
```

Farzedelim ki Selim Çarkıfelek televizyon programı için bir yazılım yapacak. Selim'den istenilen; ana bilgisayar'ın başındaki kişinin girdiği cümlelerin harf sayısını bildirmesi ve bilgisayar başındaki kullanıcıya güzelce bildirmesi. Şimdi Selime yardım edelim:

```
1
2 #! /usr/bin/env python3
3
4 #####
5 # Program Selim BAŞINDANKAYNARSULARDÖKÜLMEZ tarafından #
6 # Çarkıfelek programı için hazırlanmıştır. #
7 #
8 #
9 # Programın amacı girilen cümlelerin karakter sayısını kullanıcıya #
10 # bildirmektir. #
11 #####
12
13
14 Açılış_Yazısı = "Lütfen Sorgulamak İstedığınız Cümleyi ya da Kelimeyi girin: "
15 girilen = input(Açılış_Yazısı)
16
17 #####Sorgulama Bölümü#####
18
19 KaçKar = len(girilen)
20
21 if KaçKar != 0:
22     print("Girdiğiniz ifade " + str(KaçKar) + "karakterden oluşuyor.")
23
24 else:
25     print("Hiçbir ifade girmediniz")
26
27 input()
```

Selim'den istenilen programı, mevcut bilgilerimizle, bu şekilde yazabiliriz. Tabi ki bu program istenileni kısmen karşılayabiliyor. Zaten amaç öğrenmek olduğundan herhangi bir problem yok.

Bu programda giriş kısmını yorum satırları ile yaptık. Önce kullanıcıdan cümleyi aldık. Daha sonra bu cümleyi nesneye iliştip işlemesi için 'len' fonksiyonuna verdik. 'len' fonksiyonunu da tekrar bir nesneye bağladık. Çünkü 'len' fonksiyonunun bize vereceği değer programın geri kalan kısmında bize lazım. Yani alınan değer hafızada tutulması gerekiyor. Bu da nesnelendirme ile mümkün.

Aslında dikkatinizi bir şeyin çekmiş olması gerek.

'KaçKar' nesnesini print fonksiyonu içinde belirtirken 'str' şeklinde bi' ifade kullandık. Bunun asıl sebebi 'len' fonksiyonunun sayı değeri döndürmesi. Yani, print ifadesine aynı anda hem sayı değeri, hemde string değeri kabul etmiyor. Biz hata almamak için 'KaçKar' nesnesini yeniden string ifadeye çevirdik. Anlatım biraz kapalı oldu farkındayım. O zaman hemen açık bir anlatıma geçelim.

7 Python’da Tip Dönüşümleri

Python’da şu anda veritipi olarak iki tip hakkında bilgi sahibiyiz. Bunlardan biri karakter dizileri (string) ve sayı değerleri.

Karakter dizilerini tırnak işareti ile belirtirken, sayı tipinde verileri belirtmek için herhangi bir özel karaktere ihtiyacımız yoktur. Bu iki tip dışında yeni bir veri tipi ise ondalık sayılardır.

Ondalık sayılar bildiğimiz matematikteki ondalık sayılardır. ‘10.4’, ‘5.3’ gibi. Burada normal bildiklerimizden farklı olan tek şey Türkçe’de bildiğimiz ondalıklı sayılar şeklinde ifade edilmediğidir; virgül yerine nokta kullanılmasıdır.

Python veritipleri ile ünlü bir programlama dilidir. Bu nedenle fazla ilerlemeden bu konu hakkındaki bilgimizi artıralım derim:

Python programlama dilinde bir verinin ne tipte olduğunu öğrenmek için ‘type’ fonksiyonu kullanılır. Burada verilen çıktı verinin tipidir. Tabi ki çıktılar ingilizcedir.

- **<class ‘int’>** : ‘type’ fonksiyonundan aldığımız bu yanıt, ona verdiğiniz nesnenin bir sayı nesnesi olduğunu belirtir.
- **<class ‘str’>** : Bu çıktı verdiğiniz nesnenin karakter dizisi (string) değeri olduğunu belirtir.
- **<class ‘float’>** : Bu çıktı ise nesnenin ondalık sayı olduğunu belirtir.

Aşağıdaki örnekleri IPython3 veya uçbirimden deneyebilirsiniz:

```
1 >>> a = 5
2 >>> b = 'DenizKızı'
3 >>> c = ' 8 Kebap'
4 >>> d = "12"
5 >>> e = 12.58
6 >>> f = "12.58"
7
8 >>> type(a)
9 <class 'int'>
10
11 >>> type(b)
12 <class 'str'>
13
14 >>> type(c)
15 <class 'str'>
16
17 >>> type(d)
18 <class 'str'>
19
20 >>> type(e)
21 <class 'float'>
22
23 >>> type(f)
24 <class 'str'>
```

Eğer genel anlamda anlaşıldıysa bu veritipleri hakkında detaylı bilgi edinelim.

7.1 Sayı Değerleri (Integer)

Python’da sayı değerleri bildiğimiz sayı değerleri ile eşdeğerdir. Yani matematik dersinde kullandığımız tam sayı değerleri. Aritmetik işlemler yapılırken kullanılabilir.

Sayı değeri oluşturmak için ek bir şey yapmaya gerek yok. Örneğin

```
1 >>> a = 2
```

ifadesinde 'a' bir sayı değeridir.

İfadeleri sayı değerine dönüştürmek için 'int' fonksiyonuna ihtiyaç duyarız.

```
1 >>> a = "3"  
2 >>> type(a)  
3 <class 'str'>
```

Şu anda tanımladığımız 'a' değeri karakter dizisi. Şimdi aynı 'a' nesnesini 'int' fonksiyonu ile sayı değerine dönüştürelim:

```
1 >>> type(int(a))  
2 <class 'int'>
```

Şu anda 'int' fonksiyonunun yaptığı işi gördük ama bu durum kalıcı olmadı. Tekrar 'type(a)' kodunu denerseniz bu nesnenin hala string ifadesi olduğunu görürsünüz. Bunun kalıcı olması için çeviri sonucunu başka bir nesneye atamalıyız. Hemen görelim:

```
1 >>> c = "123456"  
2 >>> d = int(c)  
3  
4 >>> type(d)  
5 <class 'int'>  
6  
7 >>> type(c)  
8 <class 'str'>
```

Tabi ki siz bu işi şöylele yapabilirsiniz. Bu şekilde bilgisayar belleğinden tasarruf edersiniz.

```
1 karakter = "34"  
2 karakter = int(karakter)
```

Bu şekilde 'karakter' nesnesini iki kez tanımladık. En son ne şekilde tanımlandıysa nesne odur. Bu örnekte 'karakter' nesnesi artık bir sayı değeridir.

7.2 Karakter Dizisi (String)

Karakter dizilerini aslında birçok yerde gördük. Yani bu tipe aşinayız. Öncelikle karakter dizilerini nasıl tanımlıyorduk bunları görelim:

```
1 >>> K_dizi_1 = "Bu Bir Karakter Dizisidir"  
2 >>> type(K_dizi_1)  
3 <class 'str'>  
4  
5 >>> K_dizi_2 = 'E Buda Bir Karakter Dizisidir'  
6 >>> type(K_dizi_2)  
7 <class 'str'>  
8  
9 >>> K_dizi_3 = """Bu Bir karakter dizisimidir?  
10 . . . Tabiki Karakter Dizisidir."""  
11 >>> type(K_dizi_3)  
12 <class 'str'>
```

Karakter dizilerini bu şekilde tanımlıyorduk. Tanımlama işinin böyle 3 seçeneği olmasının nedeni ihtiyaçlardı. Örneğin şu ifadeyi tanımlayabilmek için;

```
1 >>> Karakter = " 'İzmir Belalısı' ile başım dertte (!) "  
2 >>> type(Karakter)  
3 <class 'str'>
```

kodlarını kullandık. İllaha 'İzmir Belalısı' ifadesini tek tırnak ile kullanacak olsak şöyle yazabilirdik:

```
1 >>> Karakter = """ 'İzmir Belalısı' ile başım dertte (!) """
```

Verileri karakter dizisine çevirmek için str() fonksiyonundan yararlanılır.

Yani elimizde bir sayı değeri varsa bu değeri 'str()' fonksiyonu sayesinde karakter dizisi haline çevirebiliriz.

```
1 >>> a = 3  
2 >>> type(a)  
3 <class 'int'>  
4  
5 >>> a = str(a)  
6 >>> type(a)  
7 <class 'str'>
```

Python yorumlayıcısı önce eşitliğin sağ tarafındaki işi yapar.

daha sonra yaptığı işlemin sonucunda çıkan değeri eşitliğin solundaki nesneye atar.

Hatırlarsanız Çarkifelek için hazırladığımız uygulamada bu fonksiyonu kullandık.

Uygulamamızı ana hatlarıyla birdaha hazırlayalım ve anlatayım.

```
1 Açılış = '''Programa Hoşgeldiniz '''  
2 Alınan = input("Lütfen araştırmak istediğiniz cümleyi girin:")  
3 Karakter_Sayısı = len(Alınan)  
4  
5 print("Girdiğiniz veri", str(Karakter_Sayısı), "karakter içeriyor")
```

Programın daha detaylı hali önceki sayfalarda anlatıldığı için biz şimdi asıl konumuza dönelim. 'len()' fonksiyonundan aldığımız değer bir sayı değeridir. 'print()' fonksiyonu ise hem sayı değerini hemde karakter dizisini aynı anda ekranda gösteremez. Biz de 'print()' e yardımcı oluyoruz ve sayı değeri olarak döndürülen değeri karakter dizisine çeviriyoruz.

Programda bizi ilgilendiren kısmın yazılmasını birkaç farklı şekilde yapabiliriz. Bunlardan işe en çok yarayanı seçip kullanmak algoritmik düşünce yeteneği gerektirir.

Burada 'Karakter_Sayısı' nesnesinin karakter dizisine dönüştürülmüş hali işimize sadece bir defa yaradığı için bu dönüştürmeyi 'print()' fonksiyonunun içinde yaptık.

Bu nesneyle daha çok uğraşmamız gerekseydi farklı bir yolla yapabilirdik. Birdahaki sayıda karakter dizilerinin metotlarını öğreneceğiz. O zaman bu programı daha farklı ve işe yarar biçimde düzenleyeceğiz.

7.3 Ondalık Sayı (Float)

İsmindende anlaşıldığı üzere ondalık sayılar üzerinde duracağız. Merak edip alıştırma yapma şansını bulmuşsanız sayı değerlerinde bazı durumlarda verinin tipi farklı oluyordu. Evet programlama işi cümle değil. Ya da bu iş benim işim değil. Ben bir örnekle anlatayım.

```
1 >>> type(5/2)  
2 <class 'float'>
```

Anlatmak istediğim buydu. Ben sayı(integer) değerlerle uğraştım ama bana gelen değer 'float'. Sebebi 5'in 2'ye bölünmesinden çıkan sonucun 2,5 olması. Python sonucu otomatik olarak float değerine çevirdi.

```
1 >>> 5 / 2  
2 2.5
```


Buradaki bir diğer hususta Python'un ondalık sayıları ayırırken Türkçe'deki gibi virgül değil nokta işareti kullanmasıdır. Zaten virgül kullanırsanız hata alacaksınız.

Bir nesneyi ondalık sayıya dönüştürürken 'float()' fonksiyonunu kullanacağız.

```
1 >>> float(3)
2 3.0
```

Eğer bir ondalık sayı nesnesinin sadece tam kısmına ihtiyaç duyuyorsanız bu nesneyi tekrar sayı değerine çevirebilirsiniz.

8 Son

Bu serimizin 2. sayısında sonuna geldik. Umarım amacıma ulaşmışımdır. Bu seri hakkında istek ve önerilerinizi forumda SUDO E-Dergi bölümünde. Python ile ilgili sorularınızı forumda Yazılım>>Programlama>>Python bölümünde belirtebilirsiniz.

9 Sorular

Soru 1:

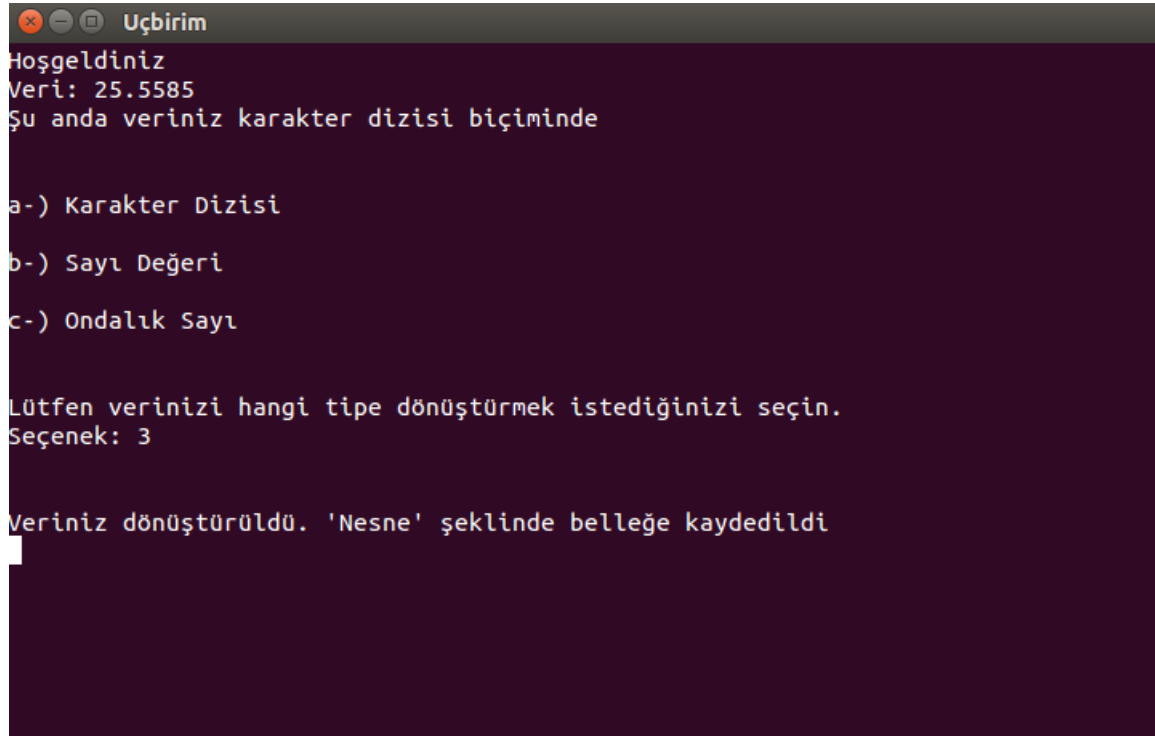
Ondalık sayıları yuvarlamak için 'round()' fonksiyonunu kullanırız. Round fonksiyonu iki parametre alır. Birincisi yuvarlanacak sayı, ikincisi noktadan sonra hangi rakamın yuvarlanacağı. Parametreler birbirinden virgül ile ayrılır. Bir örnek verelim:

```
1 >>> round(125.563 , 2)
2 125.56
3
4 >>> a = 12.8
5 >>> b = 0
6 >>> round(a, b)
7 13.0
```

```
1 Sizden herhangi bir IDE ile sayıların istenilen basamağından yuvarlanmasını sağlayan biz
   program yapmanızı istiyoruz. Program önce kullanıcıdan sayıyı almalı. Daha sonra
   kullanıcıdan hangi basamağın yuvarlanacağını almalı. Daha sonra yuvarlayıp kullanıcıya
   değeri göstermeli. Eğer yuvarlanacak taban negatif bir sayı ise kullanıcıya bu işi
   yapmayacağını belirten bir mesaj vermeli.
2
3 _İpucu_: 'round' fonksiyonu sadece ondalıklı sayı ve sayı değeri alabilir.
```

Soru 2:

Bir program yapma programı yaptığınızı ve şu an bu programda veritipleri ile uğraştığınızı düşünün. Kullanıcı önce veriyi girecek sonra bir menü ile hangi tipe dönüştürmek istediği sorulacak istenilen tipe dönüştürülecek ve kullanıcıya bildirilecek. Yani program aşağı yukarı böyle olacak:



```
Uçbirim
Hoşgeldiniz
Veri: 25.5585
Şu anda veriniz karakter dizisi biçiminde

a-) Karakter Dizisi
b-) Sayı Değeri
c-) Ondalık Sayı

Lütfen verinizi hangi tipe dönüştürmek istediğinizi seçin.
Seçenek: 3

Veriniz dönüştürüldü. 'Nesne' şeklinde belleğe kaydedildi
```

Şekil 10:

Soruların çözümlerini tartışmak, çözümlerin nasıl hazırlanacağı hakkında fikir sahibi olmanız için sizi foruma bekliyoruz...