

Çocuklar için Python ile Programlama

Buğra AYAN

Ankara 2018



**Bu eseri üç şarta sadık kalarak kopyalayabilirsiniz.
İlki, eserin tüm kopyalarında eserin ilk sahibinin belirtilmesidir.
İkincisi, eserin hiçbir kopyası veya eserden üretilmiş yeni eserlerin hiçbirisinin ticari ortamda kullanılmamasıdır. Üçüncüsü, esere dokunulmaması ve özgünlüğünün korunmasıdır.**

Kitap Versiyon 0.7

İçindekiler

Bölüm 1: Programlama öğrensek mi, öğrenmesek mi ?	3
Isırmayan yılan ile tanışın: Python.....	3
Nasıl Öğreneceğiz	4
Bölüm 2: Python Kurulumu.....	4
Bölüm 3 : Veri Tipleri.....	10
Değişken nedir ?	11
Sayılar	12
Stringler	13
Listeler	14
Tuple (Demetler)	15
Sözlükler (dictionary).....	16
Kümeler	16
Alıştırmalar	17
Bölüm 4 : Matematik Kütüphanesi'ne Giriyoruz	18
Bölüm 5: Stringler	23
Bölüm 6 : Listeler	31
Bölüm 7 : Demetler	38
Bölüm 8 : Sözlükler	44
Bölüm 9 : Fonksiyonlar.....	51
Bölüm 10 : Operatörler.....	56
Aritmetik Operatörler	57
Karşılaştırma Operatörleri.....	57
Atama Operatörleri	58
Mantıksal Operatörler	59
Üyelik Operatörü	59
Kimlik Operatörleri	60
Bölüm 11 : Koşullu Durumlar.....	60
Bölüm 12 : Döngüler	64
Bölüm 13: Modüller	70
Bölüm 14: Dosya İşlemleri	79
Bölüm 15: Python Orkestrası	85
Örnek Uygulamalar.....	85
Üçgende Kenar Hesaplama	86
Akıllı Termometre.....	86

Metin Temizleyici	87
Duygu Analiz Programı	89
Palindrom Yakalayıcı	91

Bölüm 1:Programlama öğrensek mi, öğrenmesek mi ?

Aklınızdaki bir soru ile başlayalım mı ? Neden programlama öğreniyoruz. Buna gerçekten ihtiyacımız var mı ? Bu soruya her zamankinden daha güçlü bir şekilde “Evet” cevabını verdiğimiz günlerden geçiyoruz. Çünkü tarih boyunca canlılar içinden sadece insanoğlunun sahip olduğu hikaye anlatma, hayal kurma gibi yeteneklere sahip olan yeni bir rakibimiz var. Robotlar ve yapay zeka her geçen gün daha da gelişiyor ve belki de bir gün insanların yapabildiği bir çok işi yaparak, bizi anlamsızlaştıracaklar.

Bir dakika, bir dakika ! Hemen karamsarlığa düşmeyelim. Şu an belki bir paragrafı dahi aklımızda zor tutuyoruz ama ABD’de 2016 yılında yapılan bir çalışma insan beyninin 4.6 milyar kitabı hafızasında tutabileceğini gösterdi. Bu müthiş değil mi ? Demek ki beynimizi yeterince iyi kullanmıyoruz. Fakat tembellik çağının sonuna doğru geliyoruz. Artık işler değişiyor.

Durum böyleyken hem bizim hem de yeni nesillerin yaratıcı, sorgulayan ve problem çözen bir düşünme yapısına sahip olması büyük bir önemli. Bunun için de yapmamız gereken şeylerin başında “algoritmik düşünebilme”, “programlama bilme” geliyor.

Evet. Dünyayı kurtarma görevi sizde, insanoğlunun geleceği size bağlı ! Ama aynı zamanda çok da eğlenceli bir görev değil mi bu ? Evrendeki bir probleme bazen meydan okuyorsunuz, bazen çözüyorsunuz. Hayatı farklılaştırıyorsunuz. Tarihe yön veriyorsunuz.

Bence tüm bunlar için başlamaya değer !

Isırmayan yılan ile tanışın: Python

İtiraf ediyorum , kızmaya hakkınız var. “Ya zaten bilmediğimiz bir dünyaya giriyoruz, gözümüz korkmuş. Bir de piton yılanını neden aklımıza getiriyorsun!” dediğinizi duyar gibiyim. Ama size söz veriyorum bu yılan ısırmayan bir yılan. Ayrıca son derece dost canlısı. İsteddiğiniz komutları en kısa yoldan yerine getiriyor. Onu tanıdıkça çok sevecek ve “Keşke daha önce tanışsaydık!” diyeceksiniz.

Yılanımız Python'un ilk özelliđi son derece hamarat olması. Verdiđiniz işleri hızlı bir şekilde yapıyor ve bir çok farklı alanda da çalışabiliyor. Ayrıca yılanımız özgürlüğüne de düşkün. Herhangi bir şirketin malı değil , açık kaynaklı. İçi dışı bir yanı.

Onu sevmenizi sağlayacak bir başka şey de çok hızlı öğrenebilmesi. Bilmediđi şeyleri öğrettiđinizde hızlıca sonuç alabiliyorsunuz.

Bu kadar övgü yeter, nazar da deđdirmeyelim deđil mi ? Bırakalım hünerlerini kendi göstereyin.

Nasıl Öğreneceđiz

Burada çıkacađımız Python öğrenme serüvenini önünüze koyulan bir orkestra puzzle'ını yapmak gibi düşünebilirsiniz. Düşünün ki hayatınızda hiç orkestra görmediniz. Müzik aletlerini bilmiyorsunuz, ne yapardınız ? Öncelikle müzik aletlerini öğrenir, ardından da orkestra düzeni içerisindeki yerlerini keşfederdiniz. Sonunda da bütün parçaları birleştirek puzzle'ı tamamlardınız.

İşte burada da ilk bölümlerde Python ile yapabileceğimiz örnekleri inceleyeceđiz. Yani müzik aletlerimizi tanıyacađız. Ardından da örnek projelerde farklı müzik aletlerini beraber kullanarak tıpkı orkestra puzzle'ında olduđu gibi bir büyük resim ortaya çıkaracađız.

Her bölümün sonunda da yılanımız size küçük sorular sorarak, dersleri iyi dinleyip dinlemediđinizi görecek. Eğer soruları bilmezseniz ısırmayacak korkmayın, Ama sizden biraz daha tekrar yapmanızı isteyebilir.

Bölüm 2: Python Kurulumu

Python kurulumu son derece basit. Python.org adresine giriyoruz. Ardından Download Python 3.6.3 diyerek adımları izliyoruz.

[About](#)[Downloads](#)[Documentation](#)[Community](#)[Success Stories](#)[News](#)[Events](#)

Download the latest version for Mac OS X

[Download Python 3.6.3](#)[Download Python 2.7.14](#)

Wondering which version to use? [Here's more about the difference between Python 2 and 3.](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Pre-releases](#)



Python Yükle



Python Yükleyiciye Hoş Geldiniz

This package will install **Python 3.6.3** for **Mac OS X 10.6 or later**.

Python for Mac OS X consists of the Python programming language interpreter, plus a set of programs to allow easy access to it for Mac OS X users including an integrated development environment **IDLE**.

NEW: There are important changes in this release regarding network security and trust certificates. Please see the ReadMe for more details.

IMPORTANT: **IDLE** and other programs using the **tkinter** graphical user interface toolkit require specific versions of the **Tcl/Tk** platform independent windowing toolkit. Visit <https://www.python.org/download/mac/cltk/> for current information on supported and recommended versions of Tcl/Tk for this version of Python and Mac OS X.

- Giriş

- Beni Oku

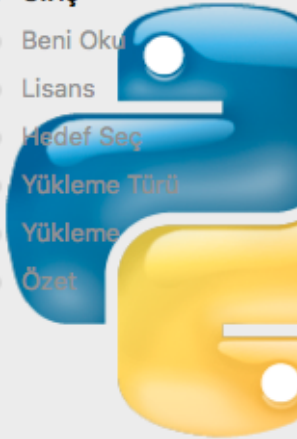
- Lisans

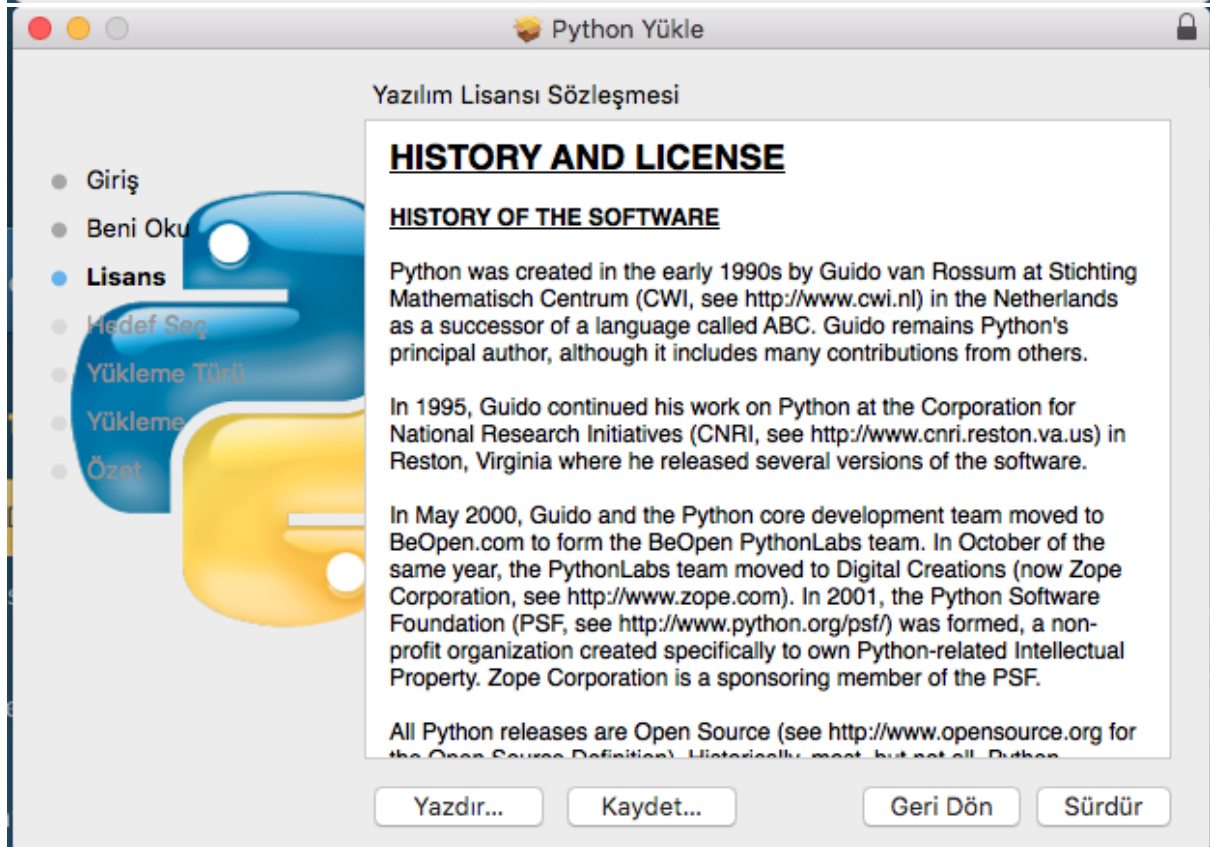
- Hedef Seç

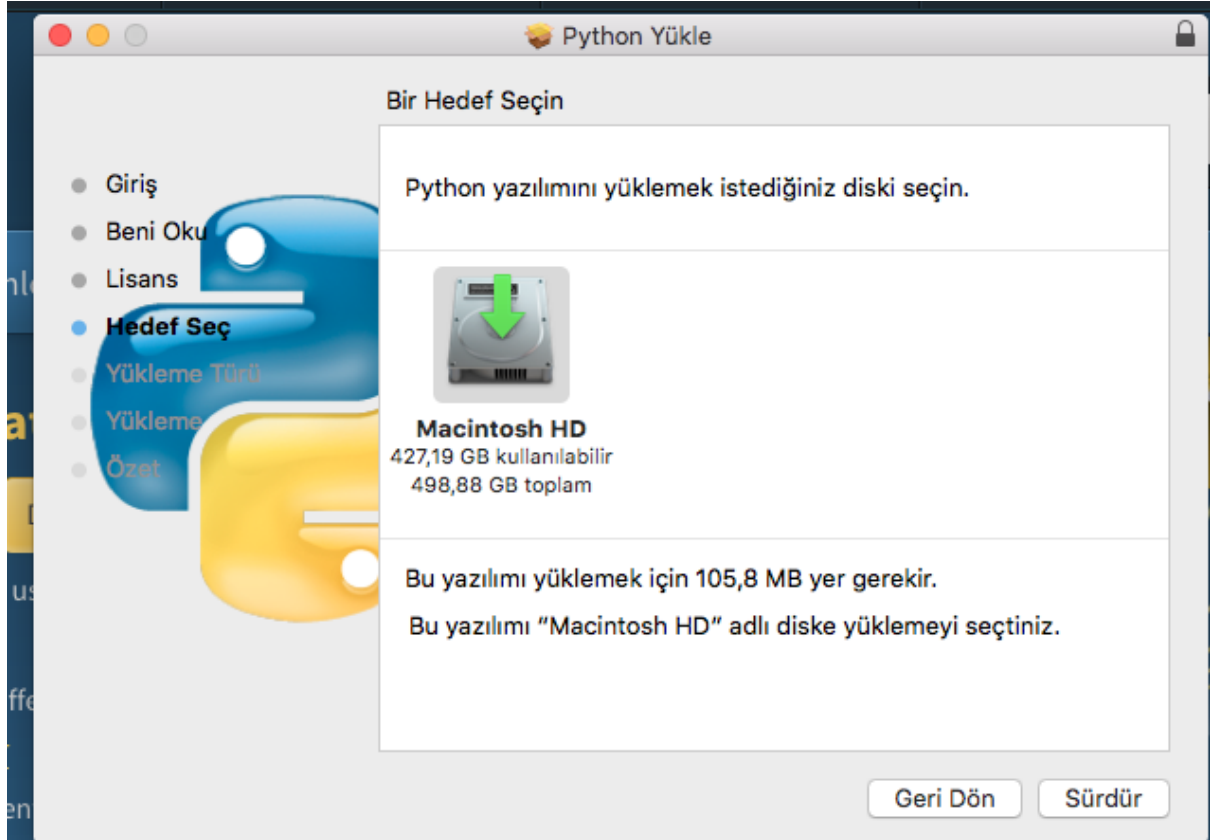
- Yükleme Türü

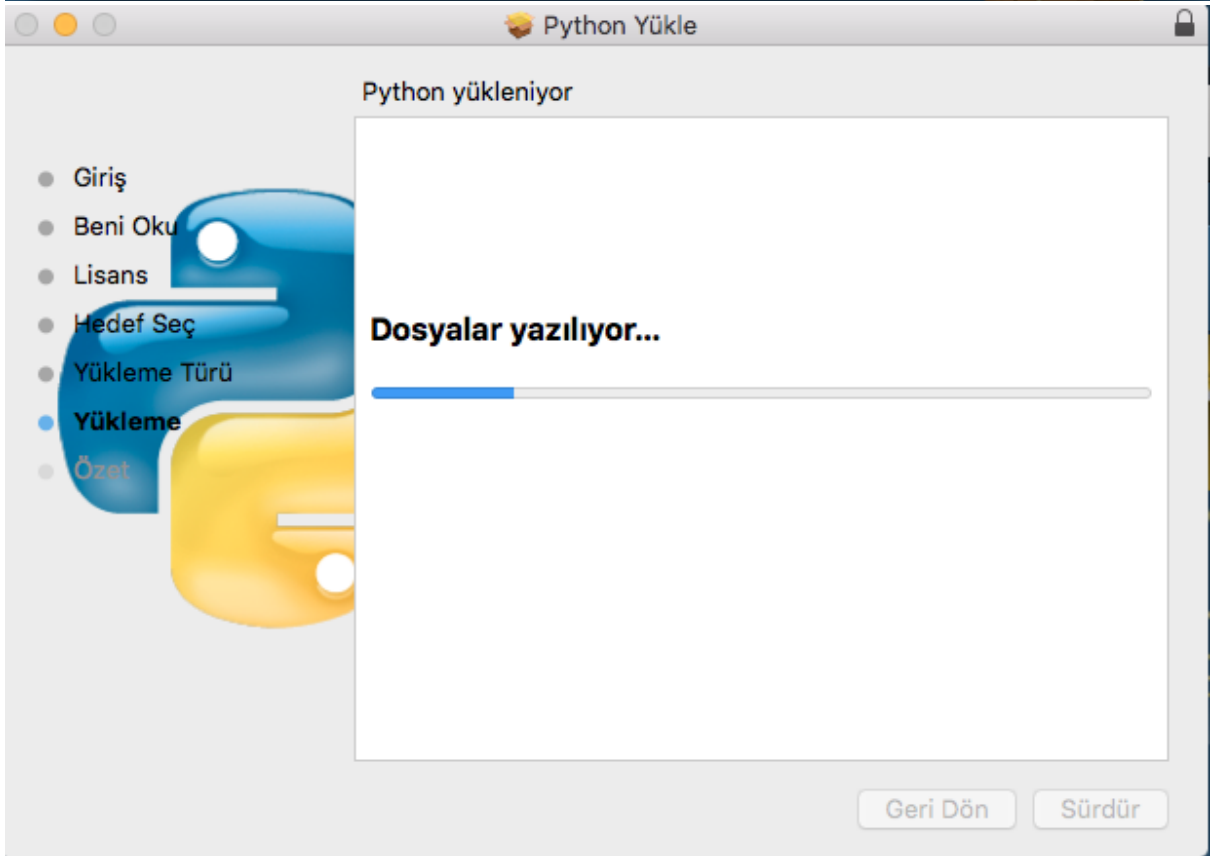
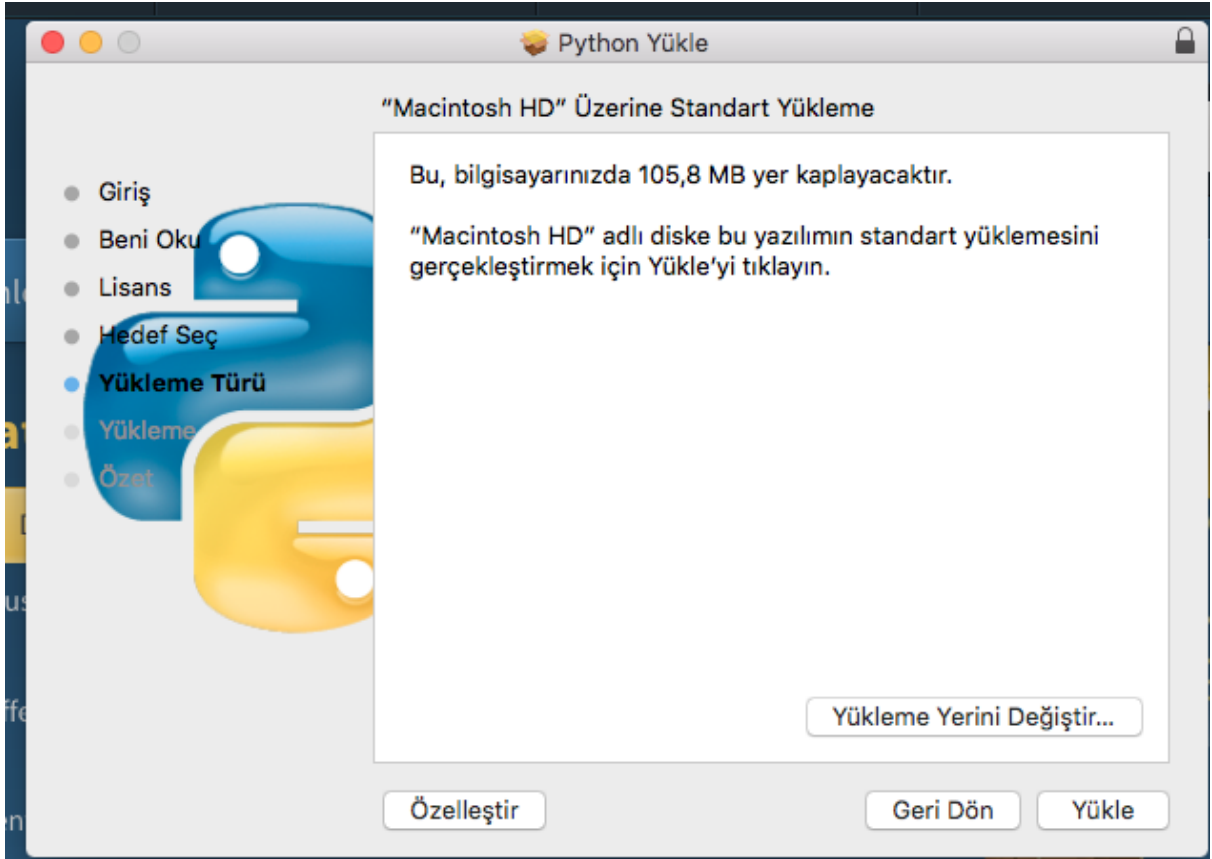
- Yükleme

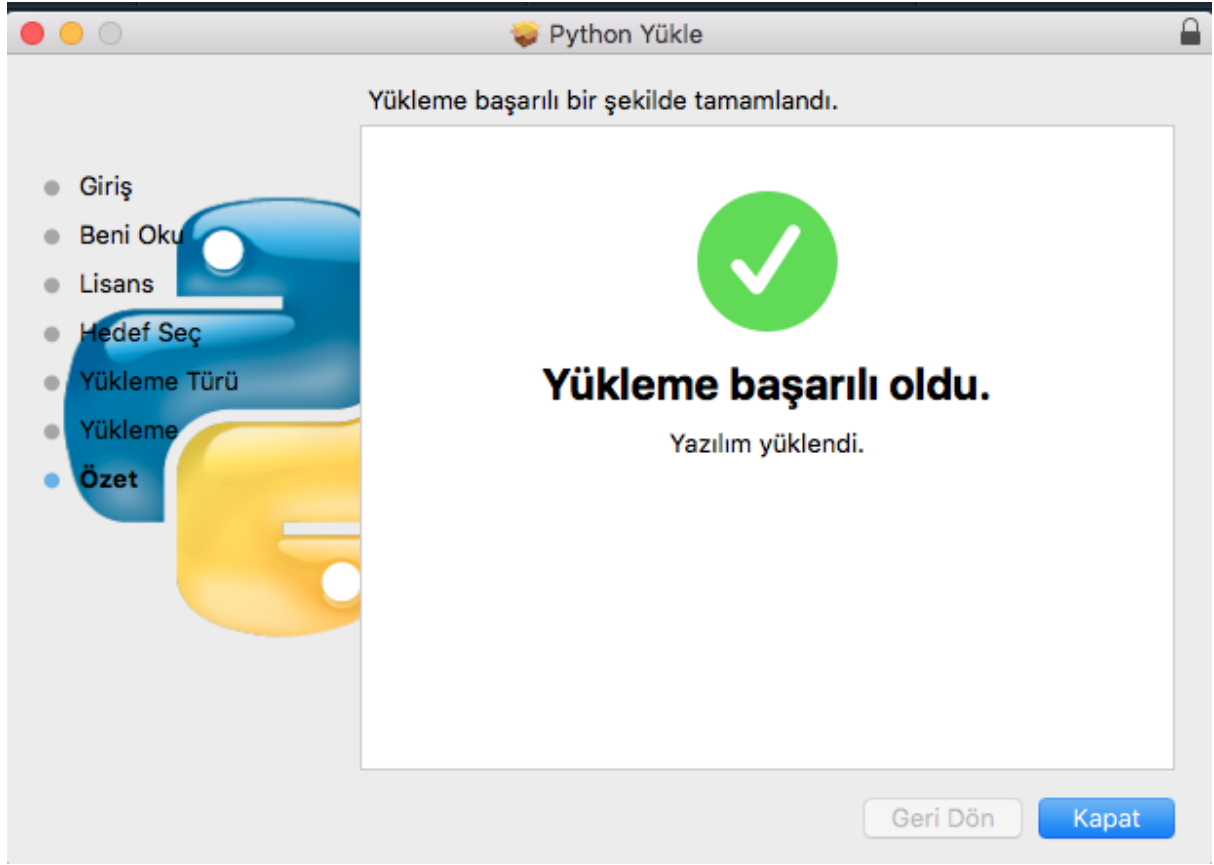
- Özet











Kişiler



Launchpad



macOS Sierra Yükle



PyCharm CE



Python 3.6



QuickTime Player



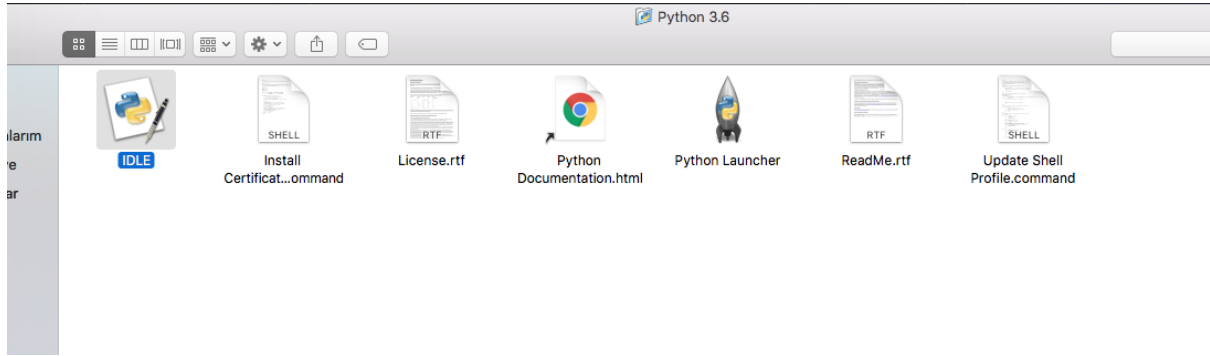
Sözlük



Takvim



TextEdit



Kurulumdan sonra IDLE dosyamıza tıklayarak Python derleyicimizi açıyoruz. Adettendir tüm programlama dillerinde ekrana “merhaba dünya” yazdırılır. Biz ise “elbet bir gün buluşacaktık” yazdırıyoruz. Burada print komutumuzu kullanıyoruz. Print yazarak ardından parantez içerisinde tırnak kullanarak metnimizi yazdırıyoruz. Sonrasında ise 3 ve 4 ü toplayarak ekrana yazdırıyoruz. Gördüğümüz gibi son derece basit. Sadece komutu giriyoruz ve saniseler içinde ekranda sonucu görüyoruz.

```
Python 3.6.3 Shell
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> print ('elbet bir gun bulusacaktik')
elbet bir gun bulusacaktik
>>> print(3+4)
7
>>> |
```

Bölüm 3 : Veri Tipleri

Birazdan Python öğrenmeye başlayacağız. Hazır mısınız ? Başlamadan önce küçük bir nokta var. Veri tiplerini ve değişken kavramımızı öğrenmemiz gerekiyor. Bunu şöyle düşünün. Kalabalık bir yolda yürüyorsunuz ve karşıdan gelen insanlarla göz göze geliyorsunuz. Eğer karşıyandan gelen kişiyi tanımiyorsanız hiç bir şey yapmıyorsunuz, uzaktan tanıyorsanız selam veriyorsunuz, yakın arkadaşınızsa durup sarılıyorsunuz. Karşıdan gelen her kişi için aslında onun “tipini” inceliyorsunuz.

- Tanımıyorsam : Yürümeme Devam Ederim
- Uzaktan Tanıyorsam: Uzaktan Selam Veririm
- Yakın Arkadaşımsa: Durup Sarılırım

Tıpkı hayatımızdaki insanların zihnimizdeki tiplerinin önemli olduğu gibi programlarda da verilerin tipleri önemlidir. Makine bir işlem yapmadan önce işlem yapacağı verinin tipine

bakar. Ona göre belleğinde yer ayırır ve işlemler yapar. Hayır ! Makineler verilere bizim gibi selam vermiyor, durup sarılmıyor ama elbette farklı davranıyor. :)

Değişken nedir ?

Şöyle düşünün. Marketten bol bol alışveriş yaptınız, eve geldiniz. Buzdolabını dolduracaksınız. Elinize bir kap alıyorsunuz ve buzdolabının üst köşesine koyuyorsunuz. Sonra bir meyve suyu şişesi alıyor ve dolabın kapağına koyuyorsunuz. Elinize her ne gelirse önce boşluklara bakıyor ardından da buzdolabına yerleştiriyorsunuz.

Programımızı yazarken elimize aldıklarımız değişkenler, onları koyduğumuz buzdolabı ise program gibidir. Program bizim değişkenlerimize göre onları hafızasına yerleştirir.

Peki neden değişkenlere ihtiyacımız var ?

Bu sorunun cevabı gayet basit. Programdan eğer bir şeyler almak istiyorsak önce ona bir şeyler öğretmemiz lazım. Tıpkı günlük hayatta olduğu gibi. Örneğin ben size iki tane sayı söylüyorum ve bunları çarpmanızı istiyorum.

Size “İlk sayı 3” dedim. Siz bunu aklınızın bir kenarında tutuyorsunuz. Ardından “İkinci sayı 5” dedim. Siz de aklınızda tuttuğunuz 3 ile 5 i çarparak 15 buldunuz. Harika ! Bir tane daha yapalım.

Şimd isize “İlk sayı 4” dedim. Artık zihninizdeki ilk sayıyı 3 e değil 4 e eşitlediniz. Ardından “İkinci sayı 6!” dedim. Bu sefer aklınızdaki ikinci sayı değişkenini değiştirmiş olduk. Adı üstünde “değişiyor” değil mi ? İşte tıpkı zihninizin işleyişi gibi programlamada da bir işleyiş var.

Biz zihnimizden yaptığımız bu alıştırmayı Python ile aşağıdaki şekilde yapıyoruz.

```
Python 3.6.3 Shell
Python 3.6.3 (v3.6.3:2c5fed86e0, Oct 3 2017, 00:32:08)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> ilksayi= 3
>>> ikincisayi=5
>>> ilksayi*ikincisayi
15
>>> |
```

Burada kısaca şunu yaptık ilksayi isminde bir değişken tanımladık ve değerini 3 olarak belirledik. Ardından ikincisayi isminde bir değişken tanımladık ve değerini 5 olarak belirledik.

Sonrasında ise bu iki değişkeni ilksayi*ikincisayi komutu ile çarparak sonuç olan 15'i ekrana yansıttık.

Sayılar

Evet, deęişken kullanımını gördük. Peki yukarıdaki buzdolabı örneğinde bahsettiğimiz gibi bu deęişkenin bir veri tipi var mı ? Bunu Python'a soralım. Komutumuz gayet basit. İngilizce'de tip anlamına gelen type ifadesini yazıyor ardından parantez açarak tipini öğrenmek istediğimiz deęişkeni yazıyoruz.

type(ilksayi) komutunu gönderdiğimizde bize class 'int' şeklinde cevap veriyor. Burada yani sınıfı: int diyor. Öğreniyoruz ki ilksayi deęişkenimizin tipi "int" miş. Int aslında integer kelimesinin kısaltması. Peki integer nedir ? Integer İngilizce'de tam sayı demek.

```
>>> ilksayi=3
>>> type(ilksayi)
<class 'int'>
>>> |
```

Biraz daha açalım mı ? Özellikle matematikle arası pek iyi olmayan arkadaşlarımız için :)

Tamsayılar olarak adlandırdığımız sayı kümesi ondalıklı olmayan sayılardan oluşur. Örneğin 3 ve 4 tam sayılara örnektir. Fakat 3,6 veya 4,84 tam sayı deęil kesirli sayıdır. Dolayısıyla bunları programımız tamsayı yani integer olarak hafızasında saklamaz. **Ne ? Bana inanmıyorsunuz demek ! Peki, hemen deneyelim.** ilksayi deęişkenimiz bu sefer 4.84 olsun bakalım sonuç ne olacak ?

```
>>> ilksayi= 4.84
>>> type(ilksayi)
<class 'float'>
>>>
```

Gördüğünüz gibi bu sefer class int deęil class float yazdı. Yani veri tipimiz deęiştı. "Peki böyle kaç tane veri tipimiz var ?" diye sorduğunuzu duyar gibiyim. **Hemen pazarlıklara başladınız valla.**

Korkulacak bir şey yok aslında. Bizim 6 tane ana veri tipi çeşidimiz var. Bunlar şu şekilde :

1. Numbers(Sayılar)
2. String(Karakter Dizileri)
3. List(Listeler)
4. Tuple(Demetler)
5. Dictionary(Sözlükler)
6. Kumeler

"İyi de ben burada int ve float'ı göremiyorum!" demeyin. Bu iki deęişken de aslında Numbers yani Sayılar deęişkenlerimizin içinde yer alıyor. Hatta Sayılar içinde yer alan bir veri tipi daha var ki o da complex olarak adlandırılan verimiz. Fakat bu karmaşık sayılar gibi çok farklı matematik hesaplarında kullanıldığı için şu an bilmenize gerek yok.

Tamam, anlaştık galiba. Sayılar içerisindeki veri tiplerini öğrendik. Peki diğer 5 tanesi ? String, Listeler , Demetler , Sozlukler ve Kumeler dediğimiz gruptaki veriler nasıl ?

Stringler

String ile başlayalım. **Acaba biz buraya kadar hiç String kullanmış olabilir miyiz ? İpucu da vereyim. Zeki Müren ? Belki de hatırladınız bile.** İlk bölümde Python derleyicimizi çalıştırdığımızda ekrana “Elbet bir gün buluşacaktık!” yazmıştık. İşte burada kullandığımız “Elbet bir gün buluşacaktık!” değişkeni String dediğimiz veri tipine giriyor. Sadece o değil aynı zamanda değişkene atamadık istediğiniz değeri ya çift tırnak(“) yada tek tırnak (') içinde yazdığınızda türü String oluyor. Genellikle harfler, kelimeler ve cümleler için kullanılıyor.

Hemen test edelim.

```
>>>
>>> ilkelime = ('tebesir')
>>> ikincikelime = ("hareketi")
>>> type(ilkelime)
<class 'str'>
>>> type(ikincikelime)
<class 'str'>
>>> ucuncukelime = ('3')
>>> type(ucuncukelime)
<class 'str'>
>>>
```

Yukarıda üç tane değişken tanımladık: ilkelime, ikincikelime ve ucuncukelime

ilkelime değişkenimizi tek tırnak içerisinde yazdık. ikincikelime değişkenimizi çift tırnak içerisinde yazdık. Ardından değişken türlerini sorduğumuzda her ikisinin de değişken türü bize str yani string olarak gösterildi. Demek ki tek tırnak veya çift tırnak kullanmamız sonucu değiştirmiyor. Bu Python dilinin bir güzelliği.

Ardından başka bir şey denedik. Önceden 3 sayısını girdiğimizde değişken tipine int demişti. Bu sefer ise 3 sayısını tırnak içerisinde bir değişken olarak girdik. Ardından tipini sorguladığımızda bize tipinin str olduğunu söyledi. Öğrendik ki tırnaklar içerisinde sayı da olsa bir kelime gibi str olarak algılanıyormuş.

Şimdi aklıma bir soru geldi. Mesela iki sayı olduğunda ve topla dendiğinde $3+5 = 8$ gibi topluyordu. Peki iki stringi topla dersek ne olur ?

Yukarıdaki örnekteki gibi değişkenlerimiz olduğunu düşünelim.

ilkelime + ikincikelime komutunu gönderdiğimizde ne sonuç alırız ?

```
>>> ilkelime = ('tebesir')
>>> ikincikelime = ("hareketi")
>>> ucuncukelime = ('3')
>>> ilkelime + ikincikelime
'tebesirhareketi'
>>> |
```

Denedik ve gördük ki ekrana iki kelimenin birleşimini yazdırıyor. Karşımıza **tebesirhareketi** yazısı çıkıyor.

Kelimeleri birleştirdiğimiz gibi bir çok şey de yapabiliyoruz. Bunları String bölümünde detaylı olarak konuşacağız. Örneğin küçük harfleri büyük harf yapabilir, belli bir harften sonrasını kesebilir, kelimeleri ters çevirebiliriz.

Şimdilik aklımızda tutmamız gereken bilgi şu :

Karakterlerden oluşan sözcük, cümle gibi şeyleri String değişkeni ile saklıyoruz.

Listeler

Bir çok şeyi aklımızda listeler ile tuttuğumuzu fark ettiniz mi ? Şimdi bile belki aklınızdan şu geçiyor. Sayı ve String değişkenlerini öğrendik. Şimdi Listeler'i öğreneceğiz. Bununla beraber 3 değişken tipini öğrenmiş olacağız.

Dikkat ederseniz bunları öğrenirken dahi aklımızda bir liste oluşturuyoruz. Listeler günlük yaşamımızda olduğu gibi Python'da da son derece önemli bir yere sahip.

Bir alışveriş programı yazdığınızı düşünelim. Alışveriş sepetine eklemeler yapıyorsunuz. Elmayı eklediniz, yoğurdu eklediniz, peçeteyi eklediniz. Liste böyle uzayıp gidiyor. Şimdi bütün bunları ayrı ayrı değişkenler olarak kaydetmek kolay olur yoksa "Alışveriş Sepetim" şeklinde bir liste tanımlayıp içine koymak mı ?

Elbette liste kullanmak işinizi kolaylaştıracaktır. İşte bunun için de Listeler değişkenleri ile sizi tanıştırmak istiyorum.

Listeler farklı veri tiplerinden değişkenleri sakladığımız veri çeşitleridir. Elemanları köşeli parantez içerisine koyar ve virgülle ayırırız. Örnek bir liste verelim.

```
alısverislistem = [ 'elma' , 'yoğurt' , 'peçete' ]
```

Burada bütün değişkenlerimiz tek tırnak içerisindeki kelimelerdi yani String türündeydi. Peki böyle olmak zorunda mı ? Elbette değil. Şöyle bir liste de olabilir.

```
benimlistem = ['elma' , 'yogurt' , 'peçete' , 3 ]
```

Şimdi listemize 3 şekilde bir dördüncü değişken ekledik. Bu değişkenimiz tırnaklar içerisinde değil. Ayrıca bir sayı. Dolayısıyla integer türünde. Listemiz sorunsuz çalıştı. Burada da gördük ki listenin içine hem String hem de Integerlar aynı anda eklenebiliyormuş. Başka değişken türleri de eklenebilir fakat şimdilik bu 2 değişken türünü öğrendiğimiz için bu örnek daha anlamlı.

Peki listeler ile neler yapabiliriz ? Biraz beyin fırtınası yapalım, neler olabilir ? Mesela bir çekiliş yapacağımız zaman kişilerin isimlerini bir listeye kaydedip ardından rastgele birini seçebiliriz.

“Tamam çekiliş güzel de şöyle günlük hayatımızdan örnek ver!” diyorsunuz sanki. Hemen söyleyeyim. Youtube kanalınızın aboneleri bir listedir, Facebook arkadaş listeniz bir listedir değil mi ?

Bu listelerde sıralama , ayırma, seçme , birleştirme , silme gibi bir çok işlem yapabiliriz.

Tuple (Demetler)

Python'da demetler diye bir değişken tipimiz var. Demetler listelere , çok benziyor. Fakat iki temel farkı var. Bunlardan birisi köşeli parantez yerine normal parantez kullanmamız ikincisi ise listenin eleman sayısının değişmemesi. Yani demetleri elemanları değiştirilemeyen listeler gibi düşünebiliriz. **Demetlerde elemanları sadece okuyabiliyoruz. Herhangi bir ekleme çıkarma silme yapamıyoruz.**

Peki bunu nerede kullanabiliriz ?

örneğin programımızda Türkiye'nin şehirlerini kullanacağız. Elimizde değişmeyecek bir liste var. Herhangi bir ekleme çıkarma silme yapmayacağız. İşte burada liste yerine tuple yani demet kullanabiliyoruz. Ayrıca tuple lar biraz sonra göreceğimiz sözlük değişken tipi için anahtar olarak kullanılabilir. Listeler ile bunu yapmak ne yazık ki mümkün değil.

```
>>> rakamlar = (0,1,2,3,4,5,6,7,8,9)
>>> rakamlar[0]
0
>>> rakamlar[8]
8
>>> rakamlar[0] = 100
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    rakamlar[0] = 100
TypeError: 'tuple' object does not support item assignment
>>>
```

Yukarıdaki örneğimizde rakamlar isminde bir demet tanımladık. Rakamları seçtik çünkü 10 tane sabit rakamımız var. Rakamlar bugün nasılsa yarın da öyle olacak değişmeyecek. Tanımlamamızın ardından print rakamlar[0] komutuyla demetin 0'inci elemanını ekrana yazdırmasını istedik. Cevap olarak 0 cevabını aldık. **Peki, “Demetin içinden başka bir eleman yazdırabiliyor muyuz?”** diye kontrol etmek için print rakamlar[8] komutu ile 8'inci

elemanını yazdırdık ve 8 cevabını aldık. harika ! demetimizi sorunsuz kullanabiliyoruz. Peki demetteki elemanları gerçekten değiştiremiyor muyuz ?

“**Valla buğra abi ben garanticiyim kontrol etmem lazım!**” diyenler vardı sanki. ben de onlar için bir komut ekledim. rakamlar[0] = 100 diyerek 0'ıncı elemanı 100 yapmak istedik. fakat görüldüğü gibi bir hata mesajı aldık. python bize bu değişikliği tuple da yapmayacağımızı söyledi.

Sözlükler (dictionary)

-'Listeler demetler derken şimdi de sözlükler mi çıktı ! İçim dışım değişken oldu Buğra abi!'

-**Tamam haklısınız biraz yorucu ama yavaş yavaş sonuna geliyoruz artık değişkenlerin. Hem bu sözlükleri çok seveceksiniz. çünkü çok işimize yarıyor. hadi biraz yakından bakalım.**

Geçtiğimiz bölümde şöyle bir şey demiştik. Türkiye'nin şehirlerini programda kullanacağız. Şehir listesi program boyunca aynı olacağı için bunu bir liste değil de demet olarak tanımlayabiliriz. Çünkü demetlerde eleman ekleme çıkarma yapamıyorduk. **Peki diyelim ki bu şehirleri plaka numaralarıyla beraber kaydetmek istiyoruz. Her şehrin bir plaka numarası var ve onlara eşleyeceğiz. Bunu listeler veya demetler ile yapabilir miyiz ? Maalesef bu mümkün değil.** İşte tam burada yeni değişkenimiz olan sözlükler devreye giriyor.

sozluk={"İzmir" : "35" , "Erzurum" : "25"} diyerek sözlüğümüzü tanımlıyoruz. Ardından ise print(sozluk["izmir"]) komutu ile izmir anahtarının değerini yazdırmasını istiyoruz. Cevap olarak 35 cevabını alıyoruz.

```
>>> sozluk = {"izmir": 35 , "erzurum" : 25 }
>>> print(sozluk["izmir"])
35
>>> |
```

Örnek sözlüğümüzü kullandık. Sözlüklere anahtar ve değerden oluşan öğeleri rahatlıkla ekleyebiliyor değiştirebiliyor ve silebiliyoruz.

Kümeler

-Kümeler mi ? Matematik mi öğreniyoruz Buğra abi ! Ne kümeleri ?

-**Ne güzel işte matematikte öğrendiğimizin aynısını programlamada kullanacağız. Ne vardı kümelerde ? Kümelerin elemanları oluyordu. Bu kümeler kesişebiliyordu birleşebiliyordu. Ayrıca bir kümenin içerisinde aynı eleman 2 kere olmuyordu. Bunları**

hatırlıyoruz değil mi ? Harika ! işte burada da aynısı var. Hadi bakalım kümeler değişkenini öğrenelim.

ilk olarak kume = () diyerek boş kümemizi oluşturuyoruz. ardından kume = set(["elma","armut","karpuz","kavun"]) komutuyla kümemizin içine elma armut karpuz ve kavun elemanlarını ekliyoruz. kümemizin içeriğini görmek için print (kume) komutuyla elemanları ekrana yazdırıyoruz.

Süper ! Peki yeni bir eleman ekleyebilir miyiz ? Bunu denemek için kume.add("muz") diyerek 4'üncü eleman olarak muz'u ekliyoruz. Tekrar kümemizi yazdırdığımızda dört eleman olduğu görülüyor. Peki listemizde olan bir elemanı tekrar eklersek ne olur ? Bunun için kume.add("elma") diyerek elma elemanını tekrar eklemek istiyoruz. Fakat yazdırdığımızda önceki durumda olduğu gibi sadece bir elma elemanı olduğu görülüyor. Oysaki değişkenimiz bir küme değilde liste olsaydı durum farklı olacaktı. Eklediğimiz her elma için listemizde yeni bir elma değişkeni tanımlanacaktı.

```
>>> kume = ()
>>> kume = set(["elma","armut","karpuz","kavun"])
>>> print (kume)
{'kavun', 'elma', 'karpuz', 'armut'}
>>>
>>> kume.add("muz")
>>> print (kume)
{'armut', 'karpuz', 'elma', 'kavun', 'muz'}
>>> kume.add("elma")
>>> print(kume)
{'armut', 'karpuz', 'elma', 'kavun', 'muz'}
>>> |
```

Süper ! Değişken tiplerimizi öğrendik. Birazcık kafanız karışmış olabilir. Fakat merak etmeyin bu değişken tiplerinin her birini metotlarıyla beraber öğreneceğiz ve bol bol üzerinden geçeceğiz.

Alıştırmalar

- 1- Öğrenci numaranı bir tam sayı olarak ve kilonu ondalıklı sayı olarak değişken tanımla. bu iki değişkeni toplayarak ekrana yazdır.
- 2- Okulunun ismini string bir değişken olarak tanımla.
- 3- İçerisinde okulunun kuruluş yılı bir öğretmenin adı ve okul numaranı olan bir liste tanımla.
- 4- İçerisinde yaşadığın yerin adı ve plaka kodu olan bir demet tanımla.

5- Sınıftan 3 sevdiğin arkadaşının isimleri ve tuttuğu takımlar olan bir sözlük oluştur. Herhangi bir arkadaşının anahtarını girerek değerini ekranda yazdır.

6- İçerisinde 5 sınıf arkadaşının olduğu bir küme tanımla. ardından bu kümeye iki arkadaşını daha add metoduyla ekle.

Bölüm 4 : Matematik Kütüphanesi'ne Giriyoruz

Bu bölümde Python ile bol bol hesap kitap işlemi yapacağız. Biraz ısınalım mı :)

Önce toplama, çıkarma, çarpma , bölme , işlemlerini birer örnekle gösterelim.

Tüm işlemlerde 12 ve 3 sayılarını kullanacağız.

```
>>> 4+21
25
>>>
>>>
>>> 12+3
15
>>> 12-3
9
>>> 12*3
36
>>> 12/3
4.0
>>> |
```

Görüldüğü gibi $12+3$, $12-3$, $12*3$ ve $12/3$ işlemlerini $+$, $-$, $*$, $/$ operatörlerini kullanarak yaptık.

İşi biraz daha zorlaştıralım mı ? Örneğin aynı anda hem çarpma hem toplama işlemi olsaydı ne olurdu ?

```
>>>
>>> 6*3+3
21
>>> 6*(3+3)
36
>>> |
```

Yukarıda iki işlem yaptık. İlkinde $6*3+3$, ikincisinde ise $6*(3+3)$ işleminin sonucunu öğrenmek istedik.

İlk işlemde herhangi bir parantez olmadığı için öncelikle çarpım işlemi gerçekleşti. $6*3=18$ oldu. Ardından buna 3 ekleyerek 21 sonucuna ulaştı. İkinci işlemde ise parantezlerden dolayı çarpma işleminden önce toplama işlemi yapıldı. $3+3=6$ olarak hesaplandı ve ardından 6 ile çarpılarak 36 sonucu ekrana yazdırıldı.

Bu örnekle de görebileceğimiz üzere tıpkı matematikte olduğu gibi Python'da yaptığımız işlemlerde de bir işlem sırası vardır. Kodumuzu çalıştırdığımızda derleyici bu işlem sırasına uyarak işlem yapar.

Benzer bir işlemi içiçe geçmiş parantezler ile de yapabiliriz.

```
>>> ((3*3)+(4*4))
25
>>>
```

Yukarıdaki işlemde içiçe parantezler kullandık. Derleyici önce iç parantezlerdeki $3*3=9$ ve $4*4=16$ işlemlerini yaptı. Ardındansa 9 ve 16'yı toplayarak 25 sonucunu ekrana yazdırdı. Peki sayıları bu şekilde doğrudan değil de bir değişkene atayarak ekrana nasıl yansıtırız ?

Şöyle bir soru düşünelim: Ali'nin bugünkü yaşı 15'tir. Alinin yaşı 5, 10 ve 15 yıl sonra kaç olur ?

Kodumuzda ilk olarak alininyasi şeklinde bir değişken tanımlıyoruz. Ardından bu değişkeni 15'e eşitliyoruz. Sonrasında ise alininyasi+5 , alininyasi+10 , alininyasi+15 diyerek ekrana yazdırıyoruz. Sonuç olarak ise 20,25,30 çıktılarını alıyoruz.

```
>>> alininyasi = 15
>>> alininyasi + 5
20
>>> alininyasi + 10
25
>>> alininyasi + 15
30
>>> |
```

Başka bir soruyla devam edelim.

Bir mobil oyunda, oyuncunun elde ettiği puan sahip olduğu altın sayısının 3 ile çarpımı, sahip olduğu yiyecek miktarının 2 ile çarpımı ve sahip olduğu can sayısının 1000 ile çarpımının toplamı şeklinde hesaplanıyor. 300 altını, 200 yiyeceği ve 3 canı olan bir oyuncunun kaç puanı vardır ?

Burada bizim 3 tane değişkenimiz var. Altın sayısı, yiyecek sayısı , can sayısı. Bu değişkenleri tanımlayalım.

```
>>> altin = 300
>>> yiyecek = 200
>>> can = 3
>>>
```

Şimdi eğer istersek puanı doğrudan ekrana yazdırırız . Veya daha şık görünmesi için puan isminde bir değişken kullanıp, print komutuyla ekrana yazdırırız. Güzel olsun diye bir de puan değişkeni tanımlıyoruz.

Neydi puanın formülü ?

Altın sayısı* 3 + Yiyecek Sayısı * 2 + Can Sayısı * 1000 . Harika ! Şimdi bunu kodda yazalım.

```
>>> altin = 300
>>> yiyecek = 200
>>> can = 3
>>> puan = (altin*3) + (yiyecek*2) + (can*1000)
>>>
```

Evet, puan değişkenimizi de tanımladık. Sırada onu print komutu ile ekrana yazdırmak var.

Heyecan büyük, geliyooor !

```
>>> altin = 300
>>> yiyecek = 200
>>> can = 3
>>> puan = (altin*3) + (yiyecek*2) + (can*1000)
>>> print(puan)
4300
>>>
```

Ve geldi. 4300 puanımız varmış. Allah bereket versin :)

Bu kadar oyun yeter, hadi kütüphaneye gidelim. Hangi kütüphane mi ? Bilmiyor musunuz bugün size matematik kütüphanesini gezdireceğim. Python içerisinde bir çok kütüphane var. Fakat bu kütüphanelerin içerisinde kitap yok. Onun yerine bizim işimizi kolaylaştıran fonksiyonlar var.

“Peki fonksiyonlar ne demek Buğra abi?”

“Fonksiyonlar bizim işimizi hızlıca yapan sihirli kelimeler. Bazen kütüphanelerdeki fonksiyonları yani sihirli kelimeleri kullanırız, bazense kendimiz bir tane oluştururuz. Aslında fonksiyonları biz oyunlardan biliyoruz. Mesela bir oyunda bazen bir şifre yazıyoruz. Bir anda canımız , enerjimiz doluyor ya, işte bu da bir fonksiyon oluyor. Bugün canımızı doldurmak için değil matematik işlemlerini hızlı yapmak için fonksiyonları kullanacağız.”

Hadi başlayalım !

İlk olarak seni karı-koca bir market işleten Celil amca ve Fulya teyze ile tanıştıracam. Celil amca çok cimri biri. Marketten ne alırsan al parayı yukarıya doğru yuvarlıyor. Mesele 3 tane gofret aldın, 2 lira 70 kuruş tuttu diyelim. Hemen yukarıya yuvarlayarak sana 3 lira diyor. Fulya teyze ise dünya tatlısı biri. Tam tersine küsuratları hiç almıyor. 3 gofret aldın 2 lira 70 kuruş tuttu diyelim yine. Fulya teyze 2 lira versen yeter diyor.

Şimdi matematik kütüphanesinde hızlıca dolaşım Celil amca ve Fulya teyzenin hangi fonksiyonları kullandığını bulalım. :)

Önce gofret=90 dedik. Yani gofretimiz 90 kuruş. Yani 0.9 TL 3 tane aldığımızda 2 lira 70 kuruş olacak. Bu da bizim ödeyeceğimiz para olsun. Para = 3*gofret dedik. Print komutu ile ekrana yazdırdık.

Şimdi öncelikle matematik kütüphanesini programımıza ekleyelim. Bunun için import math komutunu kullanıyoruz. Ardından Celil amcanın fonksiyonu ceil 'in ve Fulya teyzenin fonksiyonu floor'un nasıl çalıştığını görelim.

```
---
>>> gofret = 0.9
>>> para = 3*gofret
>>> print(para)
2.7
>>> import math
>>> math.ceil(para)
3
>>> math.floor(para)
2
>>> |
```

Görüldüğü gibi math.ceil fonksiyonu 2.7'yi 3 yaparken, floor fonksiyonu 2.7'yi 2'ye yuvarladı.

Devam edelim.

Bölme işlemi yaptık, tamam güzel. Peki kalan nasıl bulacağız ? İşte bunun için de matematik kütüphanemizdeki fmod işlemini kullanıyoruz.

math.fmod(bolunen sayı,bölen sayı) kodunu çalıştırdığımızda bu bölünme işleminden çıkan kalan ekranda yazıyor.

```
>>> math.fmod(30,3)
0.0
>>> math.fmod(30,4)
2.0
>>> math.fmod(30,7)
2.0
>>> |
```

Yukarıdaki örneğimizde 30 sayısının sırasıyla 3,4 ve 7'ye bölümünden kalanları ekrana yazdırdık. Sonuç olarak 0, 2 ve 2 değerlerine ulaştık.

Güzel gidiyoruz.

Bir de matematikte sürekli karşımıza çıkan üslü sayılar ve kareköklü sayılar var. Python'da üslü sayı ve kareköklü sayı işlemlerinin nasıl yapıldığını merak ediyor musunuz ? Hadi bakalım.

İlk olarak üslü sayılar ile başlayalım. Üslü sayılardaki mantık bir sayının, üstte yazıldığı kadar kendisiyle çarpılmasıdır. Yani eğer 2 üzeri 3 işlemi sorulursa $2*2*2$ diyerek 8 bulunur.

Python'da da bir üslü sayı işlemi yapmak istediğimizde iki tane sayı girmemiz gerekecek.

Bunlardan biri hangi sayının üssü alınacağı, diğeri ise bu sayının kaçınıcı dereceden üstü alınacağı olacak.

Bunun için de yine matematik kütüphanesinden pow adında bir fonksiyonu kullanıyoruz.

Örneğin 2 üzeri 3 işlemi için;

math.pow(2,3) dememiz yeterli.

```
>>> math.pow(2,3)
8.0
>>> math.pow(3,2)
9.0
>>> math.pow(3,3)
27.0
>>>
```

Yukarıda görüldüğü gibi 2 üzeri 3, 3 üzeri 2 ve 3 üzeri 3 işlemlerinin sonucunu bu fonksiyon ile ekrana yazdırdık.

Sıra kareköklü sayılarda. Kareköklü sayılarda bir sayının hangi sayının karesi olduğunu buluruz. Örneğin 9 , 3'ün karesi olduğu için $\sqrt{9}$ bize 3 sonucunu verir. Python'da da matematik kütüphanesindeki sqrt isimli bir fonksiyon ile sayıların kareköklerini bulabiliyoruz.

```
>>> math.sqrt(9)
3.0
>>> |
```

Yukarıda görüldüğü gibi küçük bir kod ile 9 sayısının karekökü olan 3 sayısını bulduk.

Yorulduk mu ? Matematik kütüphanesinin içindeki gezintimizi yavaş yavaş bitiriyoruz.

Unutmadan söyleyeyim. Matematik kütüphanesinde fonksiyonların yanında 2 tane de sabit var. Bunlar ünlü pi ve e sayısı.

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

Yukarıda görüldüğü gibi `math.pi` ve `math.e` komutuyla bu sayıları ekrana yazabiliyoruz.

Şimdilik bu kadar yeter. Ama eğer, “Buğra abi ben sonra yine gelir bu matematik kütüphanesini gezerim, içinde başka neler var?” diyorsanız bir dahaki gelişinizde trigonometri ve logaritma fonksiyonlarına da göz atmanızı tavsiye ederim.

Bölüm 5: Stringler

Bu bölümde daha önceden ismi geçen bir değişken türünü , “String” leri daha yakından tanıyacağız. İçerisinde harfler ve rakamlar bulundurabilen Stringler ile programlamada bir çok işlem yapıyoruz.

Daha önceden nerede String kullandığımızı hatırlıyor musunuz ? Mesela tırnaklar içerisinde “Elbet bir gün kavuşacaktık” şeklinde bir ifadeyi ekrana yazdırmıştık. İşte bu cümle harflerden oluşan bir String oluyor. Elbette Stringler sadece harflerden oluşuyor diyemeyiz. Tırnakların içerisine “2017” şeklinde bir değişken yazsak da bu bir String oluyor. Peki tek tırnak mı kullanıyoruz yoksa çift tırnak mı ? Python bu konuda son derece anlayışlı, bir iki hatta üç tırnak dahi kullanarak String değişkenimizi tanımlayabiliyoruz.

```
...
>>> tektirnak = 'deneme'
>>> cifttirnak = "deneme"
>>> uctirnak = '''deneme'''
>>> tektirnak
'deneme'
>>> cifttirnak
'deneme'
>>> uctirnak
'deneme'
>>> |
```

Yukarıda görüldüğü gibi her üç kullanımda da değişkenlerimiz sorunsuz tanımlanıyor.

Hadi biraz kelimelerle oyun oynayalım :)

Sayıları toplayabiliyorduk. Yani $3+5$ dediğimizde Python bize 8 diyordu. Peki “Tebesir” + “Hareketi” dersek ekrana ne yazacak hemen deneyelim.

```
>>>
>>> 'Tebesir' + 'Hareketi'
'TebesirHareketi'
>>>
```

TebesirHareketi yazdı. Peki ya çarpma işlemi ? Sayılarda 3*5 dediğimizde ekrana 15 yazıyordu. Peki Stringlerde ne oluyor ? Örneğin 'Tebesir' * 5 dersek ekranda ne çıktı olur ?

Deneyelim.

```
>>>
>>> 'Tebesir' * 5
'TebesirTebesirTebesirTebesirTebesir'
>>> |
```

TebesirTebesirTebesirTebesirTebesir sonucunu ekranda gördük.

Evet. Stringleri sayılar gibi toplayabiliyor veya çarpabiliyoruz. Peki sırada ne var ? Stringler ile ilgili bir çok özelliğe bakacağız. Onları büyütme , küçültme, belli bir sıradaki harfi bulma, belli bir kısmını ekrana yazdırma gibi. Fakat bunların hepsinden önce değişkenimizi tanımlamamız gerekiyor.

Ben soyad diye bir değişken tanımlayıp bunun değerini ayan olarak belirliyorum. Siz de kendi soyadınız için ayanısını yapabilirsiniz.

Yani; soyad = 'ayan' diyerek tanımlamayı yapıyoruz. Artık soyad değişkeni üzerinde yaptığım tüm işlemler ayan ifadesi için uygulanacak.

İlk işlemim basit bir uzunluk hesabı. Bize len fonksiyonu yardımcı oluyor.

len(soyad) dediğimde 4 sonucu ekranda yazıyor. 'Ayan' ifadesi 4 harften oluştuğu için 4 sonucu geldi. Eğer soyadım "Yılmaz" olsaydı burada 6 sonucunu görecektik. Uzunluğu bulmayı öğrendik. Sırada "indis" ler üzerinden işlem yapma var.

"Buğra abi indis ne demek?" dedi sanki birisi. İndis String içerisindeki her harf veya rakamın sıra numarası. İndis numarası soldan sağa doğru 0'dan başlar ve birer birer artar.

Örneğin ayan ifadesi için indis numaraları aşağıdaki gibi oluşur.

a	y	a	n
↓	↓	↓	↓
0	1	2	3

Eğer biz `soyad[0]` dersek Python şunu yapar. Soyad değişkenimize bakar ve değerini "ayan" olduğunu görür. Ardından ayan değerinin sıfırıncı indisi olan a harfini ekrana yazdırır. Aynı şekilde `soyad[1]` dediğimizde y `soyad[2]` dediğimizde a ve `soyad[3]` dediğimizde n olarak sonuç verdiğini görebiliriz.

```
>>> soyad = 'ayan'
>>> soyad[0]
'a'
>>> soyad[1]
'y'
>>> soyad[2]
'a'
>>> soyad[3]
'n'
>>> |
```

Peki indisleri sadece soldan sağa doğru mu yazıyoruz. Diyelim ki elimizde çok uzun bir ifade var. Örneğin: `tekerleme = 'bir berber bir berbere gel beraber bir berber dükkkanı açalım demis'` şeklinde bir değişken tanımladık. Sadece son harfi ekrana yazdırmak istiyoruz. Burada tek tek tüm harfleri saymak yerine Python bize bir kolaylık sağlıyor. İndisleri sağdan sola doğru yazdırırken -1, -2, gibi numaralar kullanabiliyoruz. Yine rahat anlayabilmek için kısa bir kelime olan ayan ifadesi üzerinden bakalım.

a	y	a	n
↓	↓	↓	↓
-4	-3	-2	-1

İndislerimizi görüldüğü gibi düşünebiliriz. `soyad[-1]` denildiğinde ekrana n, `soyad[-2]` denildiğinde a, `soyad[-3]` denildiğinde y, `soyad[-4]` denildiğinde a yazacaktır.

Hadi deneyelim.

```
>>>
>>> soyad[-1]
'n'
>>> soyad[-2]
'a'
>>> soyad[-3]
'y'
>>> soyad[-4]
'a'
>>> |
```

Beklediğimiz sonuçları aldık. Şimdi tekerlememize dönelim. En son harfini ve onun bir öncesindeki harfi ekrana yazdıralım.

```
-  
>>> tekerleme = 'bir berber bir berbere gel beraber berber dukkani acalim demis'  
>>> tekerleme[-1]  
's'  
>>> tekerleme[-2]  
'i'  
>>>
```

Ln: 403 Col: 4

Sonuç olarak s ve i harfi çıktılarını aldık.

Peki indisler ile sadece bir harfi mi yazdırabiliriz ? Elbette hayır. Bunun çok daha fazlasını yapabiliriz. Bunun için üst üste iki nokta kullanmamız yeterli.

soyad[1:] koduyla soyad değişkenimizde 1. indisten başlayarak tamamını ekrana yazdırabiliyoruz. Sonucumuz: "yan" oluyor. 0. indis olan a harfi sonuç ekranında yer amıyor.

soyad[1:3] koduyla ise 1. indisten başlayıp 3. indise kadar yazdırabiliyoruz. Unutmayın, 3. indis ekranda olmuyor. Sonucumuz "ya" oluyor.

Hemen deneyelim.

```
>>>  
>>> soyad[1:]  
'yan'  
>>> soyad[1:3]  
'ya'  
>>>
```

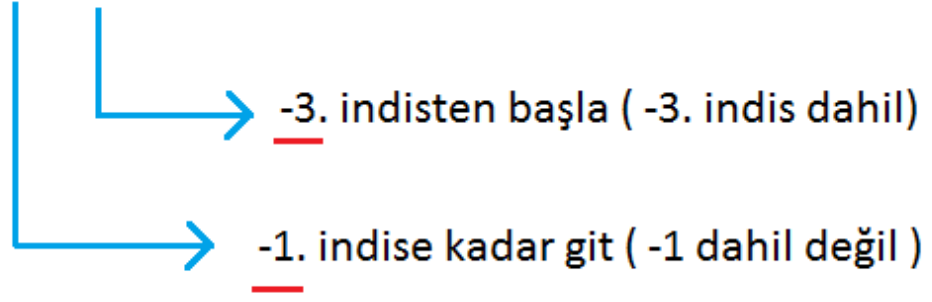
soyad[1 : 3]

3. indise kadar git. (3 hariç)

1. indisten başla. (1 dahil)

Bu işlemi soldan sağa uyguladığımız gibi sağdan sola da uygulayabiliyoruz. Burada az önce bahsi geçen -1, -2 gibi indis tanımlamalarını kullanıyoruz.

soyad [-3 , - 1]



Aşağıdaki örneğimizde ayan değerimizin -3. indisinden başlayarak -1. indise kadar giden bir kod çalıştırdık. Sonuç olarak “ya” çıktısını aldık.

```
>>>
>>> soyad[-3:-1]
'ya'
>>> |
```

Peki sadece son harfi almak istersek ne yapıyoruz ? Son derece basit. İki noktamızın öncesine bir şey yazmıyor ve sonrasında -1 diyoruz. Yani kodumuz : **soyad[:-1]**

İndislerle fazlaca uğraştık. Şimdi farklı şeyler deneyelim. Örneğin “Bir String’in içerisinde başka bir String kaç tane var ?” sorusunun cevabını bulmaya çalışalım.

```
>>> soyad.count("y")
1
>>> |
```

soyad değişkenimiz ayan olarak tanımlanmıştı. Bunun içerisindeki y harfini count fonksiyonu ile aratalım.

soyad.count(“y”) dediğimizde 1 sonucunu alıyoruz.

Çünkü y harfi 1 tane var. soyad.count(“a”) dediğimizde ise 2 cevabını alıyoruz.

Bir başka fonksiyonumuz ise upper. Upper fonksiyonu String içerisindeki bütün harfleri büyük harfe çeviriyor.

soyad.upper() kodu ayan ifadesini AYAN yaparken, soyad.lower() kodu AYAN ifadesini ayan olarak değiştiriyor. Aynı şekilde büyük ve küçük harflerden oluşan bir AyAn ifadesi de fonksiyonlara girerek tamamen büyük veya tamamen küçük harfe dönüşebiliyor.

```
>>> soyad.upper()
'AYAN'
>>> soyad.lower()
'ayan'
>>> |
```

Peki bir String'in belli bir ifadeyle başlayıp başlamadığını nasıl kontrol ederiz? Burada startswith fonksiyonumuz devreye giriyor.

soyad.startswith("a") kodunu girdiğimizde true yazıyor. Yani, "Evet ifade a ile başlıyor." diyor. soyad.startswith("Prof.") dediğimizde ise sonuç false oluyor. Yani Python bize "Hayır bu ifade Prof. ile başlamıyor." şeklinde dönüş yapıyor.

```
>>>
>>> soyad.startswith('a')
True
>>> soyad.startswith('Prof.')
False
>>> |
```

Harika ! Artık Stringler için bir başlangıç kontrolü yapabiliyoruz. Devam edelim. Bir String içerisindeki bir başka Stringi bulabilir miyiz ?

Bunun için 2 fonksiyonumuz var. İlki index.ikincisi ise find. Şimdi bu fonksiyonları sırayla kullanalım ve farklarını görelim. Her iki fonksiyonda da ayan kelimesi içerisinde "a" ve "r" harflerini aratacağız.

soyad.index("a") ve soyad.find("a") dediğimizde aynı cevabı, 0 cevabını alıyoruz. Çünkü a harfi ilk defa 0. indiste yer alıyor. Fakat "r" harfini arattığımızda soyad.index("r") kodu hata dönderirken soyad.find("r") kodu -1 sonucunu dönderiyor.

```
>>>
>>> soyad.index("a")
0
>>> soyad.find("a")
0
>>> soyad.index("r")
Traceback (most recent call last):
  File "<pyshell#253>", line 1, in <module>
    soyad.index("r")
ValueError: substring not found
>>> soyad.find("r")
-1
>>> |
```

Bunun sebebi şu. Index fonksiyonu sadece harfin veya istenen ifadenin string içerisinde olma durumunda çalışırken , find fonksiyonu harfin veya istenen ifadenin olmaması durumunda da -1 değerini dönderiyor ve çalışıyor. Bu açıdan find , index e göre daha üstün bir fonksiyon olarak görülebilir.

Yorulduk mu ? Hayırr !

Devam ediyoruz, bir yerlere kaçmak yok.

Şimdi sırada replace fonksiyonu var. Mesela Ankara'lı biri sisteme yaşadığı şehri girerken Angara yazmış. Sizin bir kod ile bunu düzeltmeniz lazım.

```
>>>
>>> sehir = 'angara'
>>> sehir.replace('g','k')
'ankara'
>>> |
```

sehir = angara diyelim. Bunu replace fonksiyonu ile düzelteceğiz.

replace fonksiyonunda iki tane bölümümüz var. İlk yere değişmesini istediğimiz ifadeyi, ikinci yere de değişince gelecek ifadeyi yazıyoruz.

Örneğin: sehir.replace("g","k") dediğimizde. Angara ifadesi içerisindeki tüm g harflerini k yapıyor. Şöyle de diyebilirdik. sehir.replace("anga", "anka") Burada da anga gördüğü her yeri anka şeklinde değiştiriyor Python.

Yavaş yavaş sona doğru geliyoruz. Bir kaç fonksiyonumuz kaldı. Stringlerin harfler veya sayılardan oluştuğunu söylemiştik. Harfler ve sayıların içinde olduğu kümeye alfanumerik ifadeler deniliyor. Ezberlemenize gerek yok .Sadece kulak aşinalığınız olması için önemli.

Bir String sadece harften veya sadece sayıdan oluşabilir. Aynı zamanda hem harflerden hem sayılardan oluşabilir.

Örneğin 3 değişken tanımlayalım.

```
string1 = ("kelime")
```

```
string2 = ("2017")
```

```
string3 = ("kelime2017")
```

Şimdi bu değişkenleri isdigit(), isalpha() ve isalnum() isimli 3 fonksiyondan geçireceğiz.

isdigit() = Tamamı sayıdan oluşuyorsa true, eğer içinde harf varsa false cevabını verir.

isalpha() = Tamamı harften oluşuyorsa true, eğer içinde sayı varsa false cevabını verir.

isalnum() = İçinde harf veya sayılar varsa true cevabını verir.

```
>>>
>>> string1 = ("kelime")
>>> string2 = ("2017")
>>> string3 = ("kelime2017")
>>>
>>> string1.isdigit()
False
>>> string1.isalpha()
True
>>> string1.isalnum()
True
>>>
>>>
>>> string2.isdigit()
True
>>> string2.isalpha()
False
>>> string2.isalnum()
True
>>>
>>>
>>> string3.isdigit()
False
>>> string3.isalpha()
False
>>> string3.isalnum()
True
>>>
```

Yukarıda görüldüğü gibi her 3 değişken için testleri yaptık ve sonuçlarımızı aldık.

Son olarak bir konuya değinmek lazım. Biz Stringlerde tırnak işaretlerini kullandığımızı söyledik fakat özel bir durumumuz var. Günlük hayatta bazı kullanımlarda kelimeleri tırnak ile ayırmamız veya alıntılarını çift tırnak içerisinde göstermemiz gerekebiliyor. Peki böyle durumlarda nasıl değişken tanımlayacağız.

Örneğin: cumle = "Babam bana "Bugra eve gel" dedi" derken tırnak içerisinde tırnak oldu. Veya 'Bugün Ankara'ya gideceğim.' derken yine tek tırnak içerisinde tek tırnak oldu. Bu sorunu nasıl aşacağız.

Çözüm gayet basit. Eğer alıntı yapacaksak kaçış karakteri olarak tanımlanan \ işaretini içeride geçen kullanıyoruz.

```
>>>
>>> cumle = ('Bugun Ankara'ya geldim')
SyntaxError: invalid syntax
>>> cumle = ('Bugun Ankara\'ya geldim')
>>> cumle
"Bugun Ankara'ya geldim"
>>> |
```

Yukarıda görüldüğü gibi sorunu rahatça çözdük.

Artık Python'da String konusuna hakimsiniz. Hadi kelimeler ile oynamaya başlayalım :)

Bölüm 6 : Listeler

Bu bölümde listeleri öğreneceğiz. Listeler normalde de bizim hayatımızın bir parçası. Marketten alınacaklar listesi, sınıftaki yaramazlık yapanlar listesi, sevdiğimiz futbol takımının oyuncularını listesi derken bir bakıyoruz ki listeler her yanda var.

Peki biz neden listeleri öğreneceğiz ? Çünkü ortak bir yerde toplanabilecek değişkenleri tıpkı bir sepete koyar gibi listeler şeklinde saklamak bizim işimizi çok ama çok kolaylaştıracak.

Listelerimizde Stringlerde olduğu gibi indeksleme yapabiliyoruz. Ayrıca listemize elemanlar ekleyebiliyor, çıkartabiliyor ve sıralayabiliyoruz. Şimdi adım adım ne yapacağımızı söyleyelim.

“Tamam iyi güzel de bu bölümde listelerle ne yapacağız Buğra abi ?” diyorsunuz. Peki anlatayım. Önce liste oluşturmayı öğreneceğiz. Ardından listenin metodlarını kullanarak ekleme, çıkarma , indeksleme gibi işlemler yapacağız. Sonrasında Stringler de olduğu gibi Listelerdeki metodları öğreneceğiz. Son olarak ise bir içiçe geçmiş liste yapacağız.

Hadi başlayalım !

Öncelikle ne listesi oluşturacağımıza karar verelim. Mesela sevdiğim meyveler için bir liste yapabilirim. Bunu şu şekilde oluşturuyorum.

```
meyve = [ "portakal" , "muz" , "çilek" ]
```

Burada ne yaptığımıza yakından bakalım. Listemin adını **meyve** olarak belirledim. Ardından köşeli parantez [işareti ile listemi açtım. Liste elemanlarım olan portakal, muz, çilek birer String olduğundan onları tırnak işareti “ içerisinde yazdım. Son olarak köşeli parantez] işaretiyle listemi kapattım. Artık içerisinde 3 değişken olan meyve isimli bir listem oldu.

Şimdi ikinci bir liste oluşturalım. Bu listede benim sevdiğim şehir, sevdiğim meyve ve sevdiğim sayı olsun. Listemi şu şekilde oluşturuyorum.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]
```

İlk listeye göre burada bir fark var. Elemanlarımdan sonucusu olan 3 ü tırnak “ işareti içerisinde yazmadım. Çünkü bir bir Integer değişken. String değil. Eğer yazıyla “üç” yazsaydım o zaman tırnak içerisinde almam gerekecekti. Demek ki hem String hem de Integer gibi farklı tür değişkenleri bir liste içerisinde kullanabiliyorum.

Peki boş bir listeyi nasıl yapabilirim ?

```
boslistem = [ ]
```

Gayet basit değil mi ? Listemizi oluşturduk ama içine eleman koymadık. "Valla Buğra abi ben garanticiyim, liste olduğuna emin olmak istiyorum!" diyorsanız hemen kontrol edelim.

`type(sevdiklerim)` dediğimizde Python bunun bize bir liste olduğunu söylüyor.

(Kod örneği)

Harika !

Peki listemizi ekrana nasıl yazdıracağız ?

`sevdiklerim = ["izmir" , "muz" , 3]` ile listeyi oluşturduktan sonra

```
print(sevdiklerim)
```

komutu ile listemizi ekrana yazdırabiliriz.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]
print (sevdiklerim)
```

```
[GCC 4.8.2] on linux
>
['izmir', 'muz', 3]
>
```

Peki kimler hatırlıyor bakalım ? Stringlerde sözcüğün indekslerini ekrana yazdırıyorduk. Örneğin `isim = "bugra"` dedikten sonra `isim[0]` dediğimizde ilk indis olan b harfi ekranda görülmüyordu. Şimdi aynısını listemizde yapacağız.

`sevdiklerim = ["izmir" , "muz" , 3]` dedikten sonra `sevdiklerim[0]` ile ilk eleman olan "izmir" 'i ekrana yazdırabiliyoruz. Aynı şekilde `sevdiklerim[1]` dediğimizde ikinci eleman olan "muz" u ekrana yazdırıyoruz. Ama eğer listemizde olmayan bir şeyi yani `sevdiklerim[30]` diyerek otuzuncu elemanı yazdırmak istersek hata verecektir.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]
print (sevdiklerim[0])
print("Şimdi olmayan eleman deneyelim.")
print (sevdiklerim[30])
```

```
[GCC 4.8.2] on linux
>
izmir
Şimdi olmayan eleman deneyelim.
Traceback (most recent call last):
  File "python", line 4, in <module>
IndexError: list index out of range
>
```

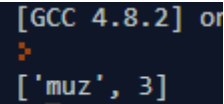
Sadece istediğimiz elemanı değil aynı zamanda belirli bir elemandan sonrasını, öncesini veya iki eleman arasındaki elemanları da ekrana yazdırabiliyoruz. Örnek olarak;

sevdiklerim[:2] dediğimizde ikinci indekse kadar olan elemanlar ekrana yazdırılır.

sevdiklerim[1:3] dediğimizde birinci indeksten üçüncü indekse kadar olan elemanlar ekrana yazılır.

sevdiklerim[2:] dediğimizde ikinci indeksten sonrası ekrana yazdırılır.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]  
print (sevdiklerim[1:3])
```



```
[GCC 4.8.2] on  
>  
['muz', 3]  
>
```

Diyelim ki bunu yazdıktan sonra aklınıza bir şey geldi. "Aaa benim sevdiklerim arasında minecraft oyunu da vardı!" listeye minecraft ı nasıl ekleriz ? Baştan oluşturup tüm elemanları tekrar yazmaya gerek var mı ? Tabi ki yok.

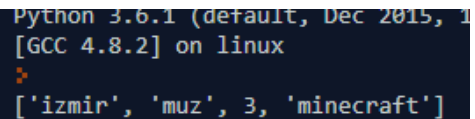
Bunun için append metodunu kullanıyoruz. Aşağıdaki kodu kullanalım.

```
sevdiklerim.append('minecraft')
```

şimdi listemizi tekrar ekrana yazdıralım.

```
print(sevdiklerim)
```

```
sevdiklerim = [ "izmir" , "muz" , 3 ]  
sevdiklerim.append("minecraft")  
print(sevdiklerim)
```



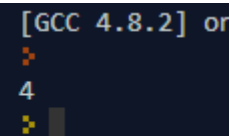
```
Python 3.6.1 (default, Dec 2015, 1  
[GCC 4.8.2] on linux  
>  
['izmir', 'muz', 3, 'minecraft']  
>
```

Gördüğümüz gibi listemiz 4 elemana ulaştı. En sonra 'minecraft' ifadesi de eklendi. Şimdi biz 4 eleman diyip aklımızdan sayıyoruz ama bu eleman sayısını kodlarla nasıl öğrenebiliriz ? Çünkü belki de 1000 elemanlı bir listemiz olacak. Oturup tek tek sayamayız.

Burada da **print(len(sevdiklerim))** komutunu kullanıyoruz. Print ekrana yazdır, len ise uzunluğu hesaplama kısmında kullanılıyor.

Deneyelim.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]  
sevdiklerim.append("minecraft")  
print(len(sevdiklerim))
```



```
[GCC 4.8.2] on  
>  
4  
>
```

Listelerle oynamaya devam edelim. Diyelim ki sevdiğim şehirlerden oluşan bir liste var. Bir de sevdiğim sayılardan oluşan bir başka liste var. Bu iki listeyi sevdiklerim diye başka bir listede toplamak istiyorum. Ne yapacağım ?

```
sehirler = [ "erzurum", "izmir" , "trabzon" ]
```

```
sayilar = [4,6,16]
```

Listeleri birleřtirmek için tek yapmam gereken **sevdiklerim = sehirler + sayilar** komutunu kullanıp **print(sevdiklerim)** demem ve birleřtirilmiř listeyi ekrana yazdırmam. Python'da iřler gerçekten kolay. İki listeyi tek hamleyle birleřtirdim.

```
sehirler=["erzurum", "izmir" , "trabzon"]  
sayilar = [4,6,16]  
sevdiklerim = sehirler + sayilar  
print(sevdiklerim)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)  
[GCC 4.8.2] on linux  
>  
['erzurum', 'izmir', 'trabzon', 4, 6, 16]  
>
```

Toplama tamam. Peki çarpma ? Bir listeyi iki katına çıkartmak için ne yapabiliriz ? Tek yapmamız gereken **carp = sehirler * 2** demek. Aynı řekilde 3 katına 5 katına da listeyi aynı řekilde çıkarabilir. Elemanları kopyalayabiliriz.

```
sehirler=["erzurum", "izmir" , "trabzon"]  
carp = sehirler * 2  
print(carp)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)  
[GCC 4.8.2] on linux  
>  
['erzurum', 'izmir', 'trabzon', 'erzurum', 'izmir', 'trabzon']  
>
```

řimdi bařka bir senaryo dűřünelim. Diyelim ki sevdiklerim = ["izmir" , "muz" , 3] řeklinde oluřtu. Fakat bir gün uyandınız ve artık ben muz sevmiyorum dediniz. Listedeki "muz" elemanını çıkartmak için ne yapacaksınız ?

Bunun için **pop** isimli metodumuzu kullanıyoruz. Listemizin ikinci elemanı olan "muz" un indeksi 1 olduđu için; **sevdiklerim.pop(1)** diyerek muz elemanını listeden çıkarıyoruz.

řimdi kodumuzu çalıřtırıp örneđi görelim.

```
sevdiklerim = [ "izmir" , "muz" , 3 ]  
  
sevdiklerim.pop(1)  
print(sevdiklerim)
```

```
[GCC 4.8.2] on  
>  
['izmir', 3]  
>
```

Hadi başka bir senaryo ile devam edelim. Diyelim ki sınıf arkadaşlarınızdan arkadaşlarım isimli bir liste oluşturduunuz. İlk olarak listeniz Ayşe ve Mehmet'ten oluşurken Ayşe sınıftan gitti ve sınıfa Betül isminde başka bir arkadaşınız geldi. Listedeki Ayşe'yi Betül ile nasıl pratik olarak değiştireceksiniz ?

İlk listemiz arkadaşlarım = ['ayşe' , 'mehmet'] şeklinde. arkadaşlarım[0] ayşe, arkadaşlarım[1] ise mehmet elemanına eşit. Şimdi ayşe yerine betül elemanı listemizde yer alacak.

Bunun için arkadaşlarım[0] = 'betül' diyoruz. Sıfırıncı indiste olan yani ilk elemanımızı betül ile değiştiriyoruz.

Listemizi print(arkadaşlarım) komutu ile tekrar yazdırabiliriz.

```
saved  
arkadaslarim = ["ayşe","mehmet"]  
print(arkadaslarim)  
print("Şimdi listemizi güncelleyelim.")  
arkadaslarim[0] = ["betül"]  
print(arkadaslarim)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)  
[GCC 4.8.2] on linux  
>  
['ayşe', 'mehmet']  
Şimdi listemizi güncelleyelim.  
[['betül'], 'mehmet']  
>
```

Peki diyelim ki sevdiğiniz şehirleri yazdınız. bugra_sehirler = ["izmir", "erzurum", "trabzon"] dediniz. Sonra Şebnem isminde bir arkadaşınız dedi ki benim de sevdiğim şehirler bunlar. Aynı listeyi nasıl kopyalayacaksınız ? Python burada işimizi kolaylaştırıyor.

sebnem_sehirler = bugra_sehirler dediğimizde sebnem_sehirler isminde bir liste oluşturuyor ve içine bugra_sehirler listesinde yer alan elemanları kopyalıyor.

```
bugra_sehirler = ["izmir","erzurum","trabzon"]  
sebnem_sehirler = bugra_sehirler  
print(sebnem_sehirler)
```

```
[GCC 4.8.2] on linux  
>  
['izmir', 'erzurum', 'trabzon']  
>
```

- Buğra abi o kadar ders var ki bazen kafamız dalgın oluyor. Bir listenin içerisinde istediğimiz elemanın olup olmadığını rahat bulabileceğimiz bir yöntem var mı?

- Hımm bir düşünelim. Belki sizin hatrınıza bir şeyler vardır, bir düşünelim. Sanırım var.

Barış Manço abimizin “Ahmet Beyin Ceketı” isimli şarkının bir dörtlüğünü String olarak alalım. Önce kelimelere ayıralım. Ardından da istediğimiz kelimenin şarkının içinde olup olmadığını kontrol edelim.

```
sarki = "Mahalleli kahvede muhabbet peşindeyken  
Leylekler lak lak edip peynir gemisi yüklerken  
Kul Ahmet erken yatar sabaha ya kısmet derdi  
Kimseler anlamazdı ya kısmet ne demektı"
```

Şarkımızı değişken olarak tanımladık. Fakat henüz ortada bir liste yok.

sarki.split() koduyla boşlukları esas alarak şarkıyı farklı elemanları olan bir listeye dönüştürelim.

```
sarki = "Mahalleli kahvede muhabbet  
kelimeler = sarki.split(" ")  
print(kelimeler)
```

```
sarki=["Mahalleli","kahvede","muhabbet","peşindeyken","  
Leylekler","lak","lak","edip","peynir","gemisi","yüklerken","  
Kul","Ahmet","erken","yatar","sabaha","ya","kısmet","derdi","  
Kimseler","anlamazdı","ya","kısmet","ne","demektı"]
```

Şimdi oluşan listemizde istediğimiz değişkeni arayalım.

ARADIĞIMDEĞİŞKEN in ARADIĞIMLİSTE şeklinde bir kod çalıştırıyoruz.

```
sarki = "Mahalleli kahvede muhabbet  
kelimeler = sarki.split(" ")  
print(kelimeler)
```

```
print("peynir" in kelimeler)  
print("zeytin" in kelimeler)
```

Örneğin;

peynir in kelimeler kodunu çalıştırdığımızda doğru anlamına gelen TRUE cevabını verirken **zeytin in kelimeler** dediğimizde yanlış anlamına gelen FALSE cevabını veriyor.

```
['Mahalleli', 'kahvede', 'muhabbet', 'peşindeyken',  
'yatar', 'sabaha', 'ya', 'kısmet', 'derdi', '', 'Kim  
True  
False  
:█
```

Evet yavaş yavaş listeler kısmının sonuna geliyoruz. Bir kaç tane daha metoda bakacağız ve ardından liste içerisine liste koymayı da konuşup konuyu bitireceğiz.

Diyelim ki elinizde sayılar isimli bir liste var ve elamanları [9,7,8,1,15] den oluşuyor. Ama size sayıların bu şekilde değil de tam dersi dizilimi lazım. Yani listenizin [15,1,8,7,9] şeklinde olmasını istiyorsunuz. Ne yapardınız ?

Python burada kolay bir yol sunuyor. sayılar.reverse() dediğinizde listenizi alıp ters çeviriyor.

```
sayılar = [ 9,7,8,1,15]  
sayılar.reverse()  
print(sayılar)
```

```
Python 3.6.1 (default, Dec  
[GCC 4.8.2] on linux  
:█  
[15, 1, 8, 7, 9]  
:█
```

“Peki Buğra abi elimizdeki sayıların büyüğünü veya en küçüğünü bize söyleyen bir metod var mı?”

“Elbette var.”

max(sayılar) en büyük elemanı min(sayılar) ise size en küçük elemanı söylüyor. Hatta listeyi sıralamanız dahi mümkün. sayılar.sort() diyerek kolaylıkla sıralayabilirsiniz.

```
sayılar = [ 9,7,8,1,15]  
print(max(sayılar))  
print(min(sayılar))
```

```
[GCC 4.8.2] on  
:█  
15  
1  
:█
```

```
sayılar = [ 9,7,8,1,15]  
sayılar.sort()  
print(sayılar)
```

```
Python 3.6.1 (default  
[GCC 4.8.2] on linux  
:█  
[1, 7, 8, 9, 15]  
:█
```

Şimdi geldik listenin içinde liste oluşturmaya. Bu kısım sona kaldı diye gözünüz korkmasın. Aslında son derece basit. Ne dedik. **Python, ısırmayan bir yılan !**

Diyelim ki bir alışveriş listesi hazırlayacağız. İçinde meyveler, sebzeler ve içecekler var. Önce 3 ayrı listenizi kaydedelim. Sonra içinde bu listeler olan 3 değişkenli bir alınacaklar listesi yapalım.

```
meyveler = [ "portakal", "üzüm", "elma" ]  
sebzeler = [ "patates", "domates", "soğan" ]  
icecekler = [ "ayran", "şalgam suyu", "limonata" ]
```

```
alinacaklar = [ meyveler, sebzeler, icecekler ]
```

print(alinacaklar) ile yeni listemizi yazdırıyoruz. Burada dikkat etmemiz gereken nokta şu. Liste içi liste tanımlarken alt listelerimizi tırnak işareti " kullanmadan üst listemize ekliyoruz. Aksi takdirde Python bunun bir liste değil de String olduğunu düşünebilecektir.

```
meyveler = [ "portakal", "üzüm", "elma" ]  
sebzeler = [ "patates", "domates", "soğan" ]  
icecekler = [ "ayran", "şalgam suyu", "limonata" ]
```

```
alinacaklar = [ meyveler, sebzeler, icecekler ]
```

```
print(alinacaklar)
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)  
[GCC 4.8.2] on linux  
▶  
[['portakal', 'üzüm', 'elma'], ['patates', 'domates', 'soğan'], ['ayran', 'şalgam suyu', 'limonata']]  
▶
```

Evet. Ekledik, çıkardık ters çevirdik, böldük , birleştirdik ve listelerin de sonuna geldik. Onlara programlarımızı yaparken bol bol ihtiyaç duyacağız. :)

Bölüm 7 : Demetler

Bu bölümde Demetler konusunu inceleyeceğiz. Demet denildiğinde aklınıza ilk ne geliyor ? Çiçekler mi ! Eğer öyleyse süper bunu aklımızda tutalım. Ankara Kızılay'da yolda çiçek satan bir Ahmet abimiz var. Ahmet abi her demete belli sayıda çiçek koyuyor. Eğer bu demete çiçek eklersek veya çıkarırsak Ahmet abi çok kızıyor. "Demetimi bozdunuz!" diyor.

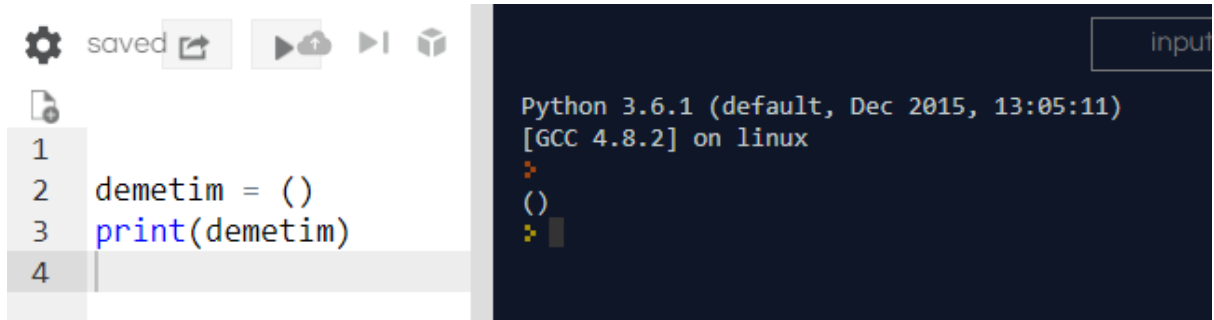
Python'ın mantığı da Ahmet abi gibi. "Eğer demet oluşturursan sonradan ekleme çıkarma yapma kardeşim!" diyor. Bunu unutmuyoruz. Demetler tıpkı listeler gibi farklı elemanlardan oluşur ama sonradan ekleme çıkarma yapılamaz.

“E ama Buğra abi listeler varken buna ne gerek var ki? Hem onda ekleme çıkarma da yapabiliyoruz!”

“Evet ama programlarımızda bazen değişmemesini istediğimiz şeyler vardır. Örneğin ülkelerin başkentlerini programın başında bir listeye kaydettik. Fakat programımızı geliştiren bir yazılımcı bu listenin elemanlarıyla oynadı ve listemizi bozdu. Bunun olmasını istemeyiz. Böyle durumlar için nasıl olsa başkentler aynı kalacak diyerek bir liste yerine demet oluşturabiliriz.

Peki bir demet nasıl oluşturulur ? Hatırlarsak liste oluştururken köşeli parantez [işareti ile listemizin elemanlarının bulunduğu bölümü açıp kapatıyorduk. Demetlerde ise köşeli parantez yerine normal parantez kullanıyoruz.

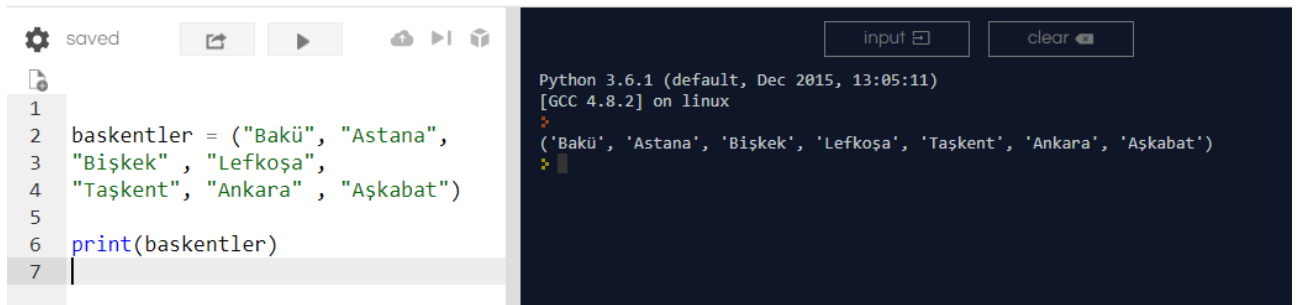
Örneğin demetim isimli boş bir demet oluşturalım ve print komutuyla ekrana yazdıralım.



```
1 demetim = ()
2 print(demetim)
```

Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
()

Şimdi içerisinde Türk devletlerinin başkentleri olan bir demet oluşturalım.



```
1 baskentler = ("Bakü", "Astana",
2 "Bişkek", "Lefkoşa",
3 "Taşkent", "Ankara", "Aşkabat")
4
5
6 print(baskentler)
```

Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
('Bakü', 'Astana', 'Bişkek', 'Lefkoşa', 'Taşkent', 'Ankara', 'Aşkabat')

Şimdi içerisinde hem String hem Integer değişken bulunan “Sevdiklerim” isimli bir demet oluşturalım. Hem demetin elemanlarını hem de türünü ekrana yazdıralım.

```
saved [share] [run] [cloud] [back] [forward] [refresh]
1
2
3 sevdiklerim = ("mavi", 4 , "izmir")
4 print(sevdiklerim)
5 print(type(sevdiklerim))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
('mavi', 4, 'izmir')
<class 'tuple'>
```

Çıktıda türümüzün “tuple” yani Türkçe karşılığıyla demet olduğunu gördük. Bir sonraki kodumuzda ise demetin içerisine bir liste değişkeni koyuyoruz ve ekrana yazdırıyoruz. İçerisinde liste olsa dahi türümüz demet olarak görülüyor.

```
saved [share] [run] [cloud] [back] [forward] [refresh]
1
2
3 demet = (["bugra", "ayan", "ankara"]
4 "elma" , 3)
5 print(demet)
6 print(type(demet))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
(['bugra', 'ayan', 'ankara'], 'elma', 3)
<class 'tuple'>
```

Şimdi geldik küçük bir problemi çözmeye. Diyelim ki ben tek bir elemanı olan demet yapmak istedim. Olur mu öyle şey ! Bal gibi olur. Python tek elemanlı bir demet de olabilir diyor. Fakat bunu tanımlarken bir String değil de demet olarak görülmesini nasıl sağlayabilirim ?

```
saved [share] [run] [cloud] [back] [forward] [refresh]
1
2
3 demet = ("bugra")
4 print(demet)
5 print(type(demet))
6
7 demet2 = ("bugra",)
8 print(demet2)
9 print(type(demet2))
10
```

```
Python 3.6.1 (default, Dec 20
[GCC 4.8.2] on linux
bugra
<class 'str'>
('bugra',)
<class 'tuple'>
```

Yukarıdaki örnekte görüldüğü gibi “bugra” isimli tek elemanı olan demetimizi yaptığımızda türümüz str yani String olarak görüldü. Fakat elemanımızdan sonra bir

virgül koyduğumuzda Python bunun türünü demet olarak gördü. Demek ki tek elemanlı demetlerde bu küçük yöntemi kullanarak demetimizi kaydedebiliyoruz.

Peki diyelim ki demetimizi kaydettik. Elemanlarına nasıl erişeceğiz ? Burada Listelerdekine benzer bir yöntem izliyoruz.

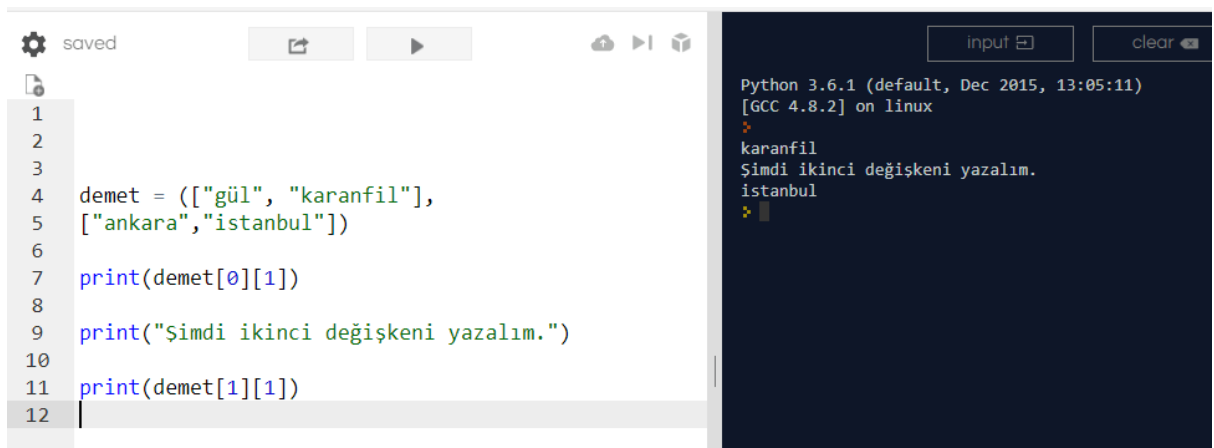
İlk örneğimizde içinde liste bulunmayan bir demeti ele alalım. İçerisinde gül, karanfil ve nergiz elemanları olan demetimizden indis numarası 1 olan yani ikinci elemene karşılık gelen "karanfil" i ekrana yazdırdık.



```
Python 3.6.1 (default, [GCC 4.8.2] on linux
>
karanfil
>
```

```
1
2
3 demet = ("gül", "karanfil", "nergiz")
4 print(demet[1])
```

İkinci örneğimizde ise içinde ikişer tane eleman olan iki listemiz var. Bu listelerin elemanlarını çağırırken çift indis kullanıyoruz. Örneğin ilk elemanımızın ilk elemanını çağırırken [0],[0] şeklinde bir kullanım olurken ikinci elemanımızın ilk elemanını çağırırken [1],[0] şeklinde kodumuzu yazıyoruz. Unutmayalım, indisler hep 0 dan başlıyor. Bu yüzden istediğimiz eleman sırasının hep bir eksiğini yazıyoruz.



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
karanfil
Şimdi ikinci değişkeni yazalım.
istanbul
>
```

```
1
2
3
4 demet = (["gül", "karanfil"],
5 ["ankara", "istanbul"])
6
7 print(demet[0][1])
8
9 print("Şimdi ikinci değişkeni yazalım.")
10
11 print(demet[1][1])
12
```

Yukarıdaki örnekte ilk önce demet[0],[1] çağrıldı. 0 kısmıyla ilk eleman olan çiçek türlerine odaklandık. Ardından [1] ile bu elemanlardan ikincisini seçtik. Ekrana karanfil yazdırdık. İkinci değişkenimizde ise demet[1],[1] kodunu kullandık. Birinci 1 kısmıyla

demetin ikinci elemanı olan şehir isimlerine odaklandık. İkinci 1 kısmıyla da bu listedeki ikinci elemanı seçtik. Ekranı "İstanbul" yazdırdık.

Peki demetlerimizle başka neler yapabiliriz ? Hadi beraber bakalım !

Benim aklıma şu geliyor. Diyelim ki elimde içinde bir çok çiçek olan bir demet geldi. Ben en çok nergizi seviyorum. Hemen içindeki nergizleri saymaya başladım. Python bunun gibi bir şey bana yapabilir mi ?

Elbette ! Bunun için count metodunu kullanıyoruz.



```
saved [share] [run] [cloud] [play] [cube]
1 demet=("Gül", "Nergiz", "Lale" , "Nergiz"
2 , "Karanfil", "Lale", "Nergiz", "Papatya")
3 print(demet.count("Nergiz"))
4
```

Python 3.6.1 (default, [GCC 4.8.2] on linux)
3

İşte bu kadar. Derleyicimiz "Nergiz" değişkenlerini saydı ve bize 3 cevabını verdi. Toplam kaç değişken olduğunu ise tıpkı listelerde olduğu gibi len fonksiyonuyla öğreniyoruz.



```
saved [share] [run] [cloud] [play] [cube]
1 demet=("Gül", "Nergiz", "Lale" , "Nergiz"
2 , "Karanfil", "Lale", "Nergiz", "Papatya")
3 print(len(demet))
4
```

Python 3.6.1 (defa [GCC 4.8.2] on lin)
8

Demetimizde toplam 8 çiçek varmış.

Şimdi aklıma bir soru takıldı. Demetimizde bir elemanın olup olmadığını nasıl kontrol ederiz ? Diyelim ki koskocaman bir liste geldi önümüze. İçerisinde bizim aradığımız elemanın olup olmadığını merak ediyoruz. Ne yaparız ? Bunun için de tıpkı listelerde olduğu gibi in ile kontrol yapıyoruz. Şimdi listemizde Açelya ve Lale nin olup olmadığını sorgulayalım.

```
saved [run] [play] [cloud] [stop] [cube]
Python 3.6.1 (default, [GCC 4.8.2] on linux)
1 demet=("Gül", "Nergiz", "Lale" , "Nergiz"
2 , "Karanfil", "Lale", "Nergiz", "Papatya")
3 print("Açelya" in demet)
4 print("Lale" in demet)
5
False
True
```

Açelya sorgumuz için False yani Yanlış cevabı dönerken Lale sorgumuz için Doğru anlamına gelen True cevabı döndü. Harika !

Benzer şekilde Listeler kısmında olan bir çok özelliği Demetler'de de kullanabiliyoruz. Fakat demetimize ekleme, çıkarma, güncelleme, silme yapamıyoruz. Peki yapmak istersek ne mi oluyor ? Hadi deneyip görelim.

```
saved [run] [play] [cloud] [stop] [cube]
Python 3.6.1 (default, Dec 2015, 13:0
[GCC 4.8.2] on linux)
1 #önce liste için deneyelim
2
3 liste = ["bugra", "ayan", "ankara"]
4 liste[2] = "izmir"
5 print(liste)
6
7 #şimdi demet için deneyelim
8 demet = ("bugra", "ayan", "ankara"
9 demet[2] = "izmir"
10 print(demet)
11
Traceback (most recent call last):
  File "python", line 9
    demet[2] = "izmir"
    ^
SyntaxError: invalid syntax
```

Görüldüğü gibi listede herhangi bir sorun olmuyor fakat demet[2] elemanını değiştirdiğimizde hata kodumuz oluşuyor.

Yavaş yavaş sona geliyoruz. Şöyle bir senaryo düşünelim. Bir liste oluşturduk fakat bundan bir tane de demet olarak saklamak istiyoruz. Listeyi hızlıca demete nasıl dönüştürebiliriz ? Bunun için DEMETİSMİMİZ = tuple(DONUSTURMEKİSTEDİGİMİZLİSTE) şeklinde bir kod kullanmamız yeterli oluyor.

```
saved [run] [stop] [refresh] [help] [exit]

1 print("Listemizi tanımlayalım.")
2
3 liste = ["bugra", "ayan", "ankara"]
4 print(type(liste))
5
6 print("Listeyi demete dönüştürelim.")
7
8 demet = tuple(liste)
9 print(type(demet))
10
```

```
Python 3.6.1 (default, Dec 2015, 13:05)
[GCC 4.8.2] on linux
>
Listemizi tanımlayalım.
<class 'list'>
Listeyi demete dönüştürelim.
<class 'tuple'>
>
```

Görüldüğü gibi liste değişkenimizin türü liste iken demet elemanımızın türü tuple yani demet olarak kaydedildi.

Bölüm 8 : Sözlükler

Isırmayan yılan Python'da bir başka kullanışlı veri tipimiz de sözlükler. Sözlük denildiğinde aklınıza ne geliyor ? Bir kelime olur ve karşısında o kelimenin anlamı olur değil mi ? İşte Python'da da bundan çok farklı değil. Bir anahtarımız olacak ve onun değeri olacak. Örneğin bir sınıfın sınav sonuçlarını kaydedeceğimizi düşünelim. Her kişinin isminin karşısında bir not olacaktır. Bunu bir sözlük yapısıyla kaydedebiliriz. Aslında günlük hayattaki bir çok şeyi sözlük dediğimiz yapıyla hafızamızda tutarız. İnsanların size olan borçlarını, lokantaların numaralarını, hangi başkentin hangi ülkeye ait olduğunu vs.

Şimdi kollarımızı sıvayıp işe koyulalım. Önce boş bir sözlük tanımlıyoruz. Sözlüğümüzün adı notlar olsun.

```
saved [run] [share] [help] [exit]

1 notlar = {}
2
3
4 print(notlar)
5
6 print(type(notlar))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{}
<class 'dict'>
```

Notlar isimli sözlüğümüzü oluştururken süslü parantez { kullandık. Print komutuyla önce kendisini sonra tipini ekrana yazdığımızda gördük ki türü dictionary yani sözlüğün kısaltması olan dict şeklinde oluştu. Eğer süslü parantez yerine köşeli veya normal parantez kullansaydık değişkenimiz liste, demet gibi farklı bir tür olacaktı.

Devam edelim. Şimdi sözlüğümüze bir şeyler ekleyelim. Adı üstünde notlar dedik. Şimdi öğrencilerin notlarını ekleyelim. Tabi ki kendime 100 vereceğim :)

```
saved [run] [share] [help] [exit]

1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Ayşe" :70
4
5 }
6
7
8 print(notlar)
9
10 print(type(notlar))
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ahmet': 50, 'Ayşe': 70}
<class 'dict'>
```

Kendime 100 notunu verdim. Ayrıca Ahmet'e 50 ve Ayşe'ye 70 verdim. Allahım ilk defa bir sınıfın en çalışkanı ben oldum. Kutlamalar başlasın :)

Aa şu işe bakın. Sınıfın haylazı Selim'in notunu girmeyi unutmuşuz. Şimdi sözlüğümüze bir ekleme yapmamız lazım. Selim sınavdan 10 almış. Bu notu ekleyeceğiz. Sözlüklere bir eleman eklemek oldukça basit. Şu şekilde kodumuzu yazıyoruz.

SOZLUGUNISMI[ANAHTAR] = DEGER yani burada notlar["Selim"] = 10 şeklinde ekleme yapacağız.

```
saved [share] [run] [refresh] [close]
1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Ayşe" :70
4
5 }
6
7 notlar["Selim"] = 10
8
9 print(notlar)
10
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ahmet': 50, 'Ayşe': 70, 'Selim': 10}
```

Görüldüğü gibi basitçe Selim'i de ekledik. Bir dakika bir dakika şu gelen Ayşe değil mi ? Yine notuna itiraz edecek galiba. Tabi sınıf birincisi olamadı ya rahat duramaz o şimdi.

- Öğretmenim ben daha yüksek bekliyordum.
- Tamam bakalım kağıdına kızım.

(10 dakika sonra)

-Ayşe sana 5 puan eksik vermişim. Notun 75 olacakmış.

Ya 5 puan için ortalığı ayağı kaldırdın Ayşe. Neyse şimdi Ayşe'nin notunu değiştirelim. Sözlüklerde güncelleme yaparken aslında eklemeye yaptığımızın aynısını yapıyoruz. SOZLUGUNISMI[ANAHTAR] = YENİDEGER örneğinin burada `notlar[Ayşe] = 75` yapacağız.

```
saved [share] [run] [refresh] [close]
1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Ayşe" :70
4
5 }
6
7 notlar["Selim"] = 10
8 notlar["Ayşe"] = 75
9
10 print(notlar)
11
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ahmet': 50, 'Ayşe': 75, 'Selim': 10}
```

Ayşe'nin notunu da 75 olarak güncelledik. Şimdi de Ahmet öğretmenin yanına gidiyor.

Ahmet-Öğretmenim ben aslında bu sınava girmemişim. Necati benim adımlı sınav kağıdına yazmış.

Öğretmen-Ne inanmıyorum ! Necati çabuk gel buraya ! Doğru mu bu ?

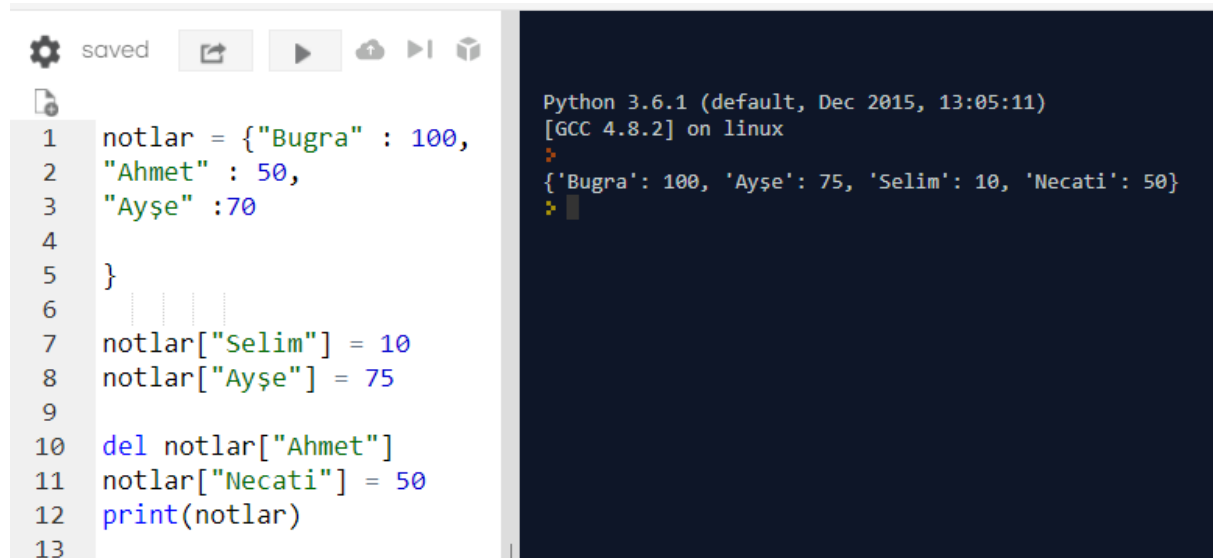
Necati-Evet öğretmenim sınavım kötü geçince Ahmet'in adını yazıp verdim.

Öğretmen- Necati bu yaptığın çok yanlış arkadaşından özür dile.

Necati- Özür dilerim Ahmet.

Ahmet- Önemli değil Necati, adımı yazıyorsun bari yüksek bir not alsaydın. :) Şaka şaka affettim ama bir daha yapma.

Şimdi bizim Ahmet'in notunu silip, Necati'ye 50 eklememiz gerekiyor. Sözlüklerde bir eleman silmek için del komutunu kullanıyoruz. Şimdi Ahmet'in notunu silelim ve Necati'ye 50 notunu ekleyelim.



```
saved [run] [stop] [refresh] [help] [exit]
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ayşe': 75, 'Selim': 10, 'Necati': 50}
>
```

```
1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Ayşe" :70
4
5 }
6
7 notlar["Selim"] = 10
8 notlar["Ayşe"] = 75
9
10 del notlar["Ahmet"]
11 notlar["Necati"] = 50
12 print(notlar)
13
```

Evet. Del komutuyla bir elemanı silebiliyoruz. Peki bütün sözlüğü sıfırlamak istersek hangi metodu kullanacağız ? Burada clear metodu bize yardımcı oluyor. Hemen deneyelim.

```
saved [share] [run] [upload] [refresh] [close]
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
{}
1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Ayşe" :70
4
5 }
6
7 notlar["Selim"] = 10
8 notlar["Ayşe"] = 75
9
10 del notlar["Ahmet"]
11 notlar["Necati"] = 50
12
13 notlar.clear()
14
15 print(notlar)
16
```

notlar.clear metodunu kullandığımızda, {} ifadesiyle ekrana boş bir sözlük yazdırdı.

Sözlüğü de sildik bölüm bitti sanmayın. Daha işimiz var. Şimdi notlar.clear() ifadesini çıkartıp sözlüğümüzle işlemler yapmaya devam ediyoruz.

Farklı örneklerle metodları kullanmayı öğrenelim. Pop metodunda istediğimiz elemanı siliyoruz. Örneğin aşağıdaki örnekte notlar.pop("Bugra") diyerek Buğra elemanımızı sildik. Del'e son derece benziyor fakat ilerleyen aşamalarda bazı küçük farklı kullanımları olabiliyor.

```
saved [share] [run] [upload] [refresh] [close]
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
{'Ahmet': 50, 'Selim': 10, 'Ayşe': 75}
1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Selim" : 10,
4 "Ayşe" :75
5
6 }
7
8 notlar.pop("Bugra")
9
10 print(notlar)
```

Popitem metoduyla devam edelim. Popitem , Pop metoduna çok benziyor fakat içerisine herhangi bir parametre almıyor. Yani doğrudan notlar.popitem() diyerek parantezin içini boş bırakıyoruz. Sözlükten rastgele bir eleman siliyor.


```
saved [run] [share] [refresh] [back] [forward] [home]

1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Selim" : 10,
4 "Ayşe" :75
5
6 }
7 notlar.popitem()
8
9 print(notlar)
10
..
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ahmet': 50, 'Selim': 10}
>
```

Örneğin yukarıdaki kodu çalıştırdığımızda Ayşe elemanını sildi.

Peki sözlüğümüzü nasıl kopyalayabiliriz. Bunun için de copy metodunu kullanıyoruz. Notlar sözlüğümüzü yeninotlar = notlar komutuyla “yeninotlar” isminde bir sözlüğe kopyalayıp bu yeni sözlüğümüzü ekrana yazdıralım.

```
saved [run] [share] [refresh] [back] [forward] [home]

1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Selim" : 10,
4 "Ayşe" :75
5
6 }
7
8 yeninotlar = notlar
9
10
11 print(yeninotlar)
12
..
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
{'Bugra': 100, 'Ahmet': 50, 'Selim': 10, 'Ayşe': 75}
>
```

İkinci sözlüğümüzü de ekrana sorunsuz bastırdık. Şimdi ise sözlüğümüzdeki anahtarları ve değerleri ayrı listeler halinde ekrana bastıralım. Yani öğrenci isimleri ve onların notlarını ayrı ayrı ekrana yazdıralım.

```
saved [run] [stop] [help] [input]

1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Selim" : 10,
4 "Ayşe" :75
5 }
6
7
8 print("Anahtarları yazdıralım")
9
10 print(notlar.keys())
11
12 print("Değerleri yazdıralım.")
13
14 print(notlar.values())
15
16 print("Anahtar ve değerleri yazdıralım.")
17
18 print(notlar.items())
19
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Anahtarları yazdıralım
dict_keys(['Bugra', 'Ahmet', 'Selim', 'Ayşe'])
Değerleri yazdıralım.
dict_values([100, 50, 10, 75])
Anahtar ve değerleri yazdıralım.
dict_items([('Bugra', 100), ('Ahmet', 50), ('Selim', 10), ('Ayşe', 75)])
>
```

Anahtarlar için keys ve değerler için values metodunu kullanarak istediğimiz içeriği ekrana bastırdık. dict_keys ile başlayan satırda bize öğrenci isimlerini, dict_values ile başlayan satırda ise öğrenci notlarını verdi. Ayrıca anahtar ve değerleri beraber ekrana yazdırmak için de notlar.items metodunu kullandık.

Şimdi bir elemanın sözlük içinde olup olmadığını kontrol edelim. Bunun için liste ve demetlerde olduğu gibi in ifadesini kullanıyoruz. Örneğin notlarda Arzu'nun notunun olup olmadığını kontrol edelim.

```
saved [run] [stop] [help] [input]

1 notlar = {"Bugra" : 100,
2 "Ahmet" : 50,
3 "Selim" : 10,
4 "Ayşe" :75
5 }
6
7
8 print("Arzu" in notlar)
9 print("Bugra" in notlar)
10
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
False
True
>
```

Görüldüğü gibi Arzu için False yani Yanlış , Bugra için True yani Doğru cevabını aldık.

Sözlükler bölümümüzün de sonuna geldik. Artık rahatlıkla sözlük değişkenleri üzerinde işlemler yapabiliyoruz.

Bölüm 9 : Fonksiyonlar

Bu bölümde fonksiyonları işleyeceğiz. Fonksiyon günlük hayatta da kullandığımız bir kavram. Mesela bir programda ilginç bir buton görünce “Buraya basınca ne oluyor?” veya “Bu butonun ne fonksiyonu var?” deriz. Araba sürerken veya ünlü bir dansı yaparken de aslında kafamızdaki önceden belirlenmiş fonksiyonları çalıştırırız. Arabaya binerken aklımızdan geçen şeyler nedir ?

Arabayı nereye park ettiğini hatırla.

Hatırlama yapıldı.....

Anahtarın hangi cebinde olduğunu hatırla.

Hatırlama yapıldı.....

Arabaya yürü.

Arabanın yanına gelindi.....

Anahtarla arabayı aç.

Araba açıldı.....

Arabaya bin ve kontağı çevir.

Kontak çevrildi.....

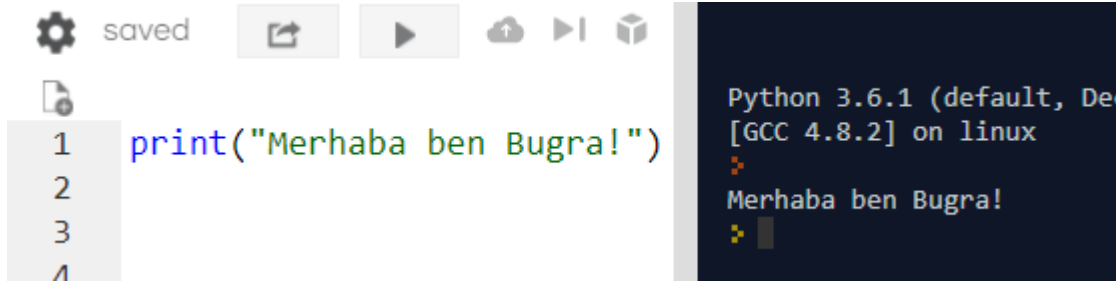
Arabayı sürmeye başla.

İşte bu olaylar aslında bir araba_sür fonksiyonu olarak zihnimizde kayıtlıdır. Aynı şekilde bir robota arama sürmeyi öğreteceğimizde de bunun gibi bir fonksiyon kullanmalıyız. Peki neden bunu fonksiyonla yapıyoruz ?

Bir düşünelim. Robotuma her araba sürdürmek istediğimde bu kodları en baştan yazmam mı daha kolay olur yoksa araba_sür diye bir fonksiyon tanımlamam mı ? Fonksiyon tanımladıktan sonra tek yapmam gereken araba süreceği zaman bu fonksiyonu çalıştırmam.

İşte bu yüzden fonksiyonlar bize programlama sırasında müthiş bir hız katarlar. Peki biz daha önce hiç fonksiyon kullandık mı ? Elbette ! Örneğin ekrana bir şey yazdırırken kullandığımız Print ve değişkenlerin tipini öğrenirken kullandığımız Type birer fonksiyondur. Bu fonksiyonlar Python tarafından oluşturulmuş fonksiyonlardır. Fakat biz de kendi fonksiyonlarımızı oluşturabiliriz.

Print fonksiyonuna tekrar yakından bakalım.



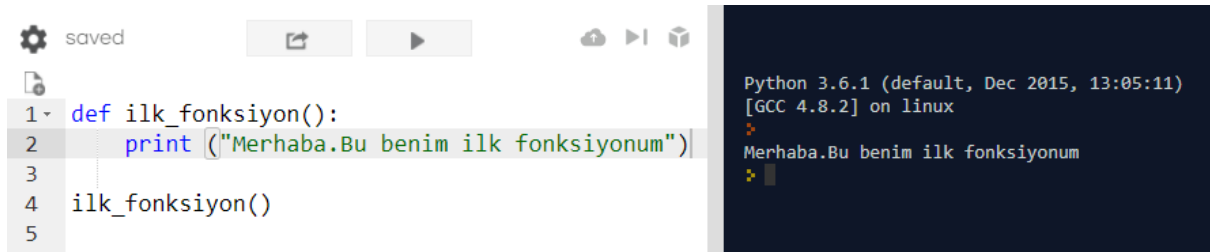
```
saved [share] [run] [cloud] [stop] [cube]
1 print("Merhaba ben Bugra!")
2
3
4
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Merhaba ben Bugra!
```

Burada print fonksiyonu **Merhaba ben Bugra** deęişkenini aldı. Fonksiyonun alıřma mantığı gereęi bu deęiřkeni doęrudan ekrana yazdırdı.

řimdi aynı iřlemi ilk_fonksiyonum ismiyle oluřturduęumuz kendi fonksiyonumuzdan faydalanarak yapalım. Bunun iin İngilizce’de tanımla anlamına gelen “define” kelimesinin kısaltması def ifadesinden faydalanıyoruz. def ilk_fonksiyon(): diyerek fonksiyonumuzu tanımladık.

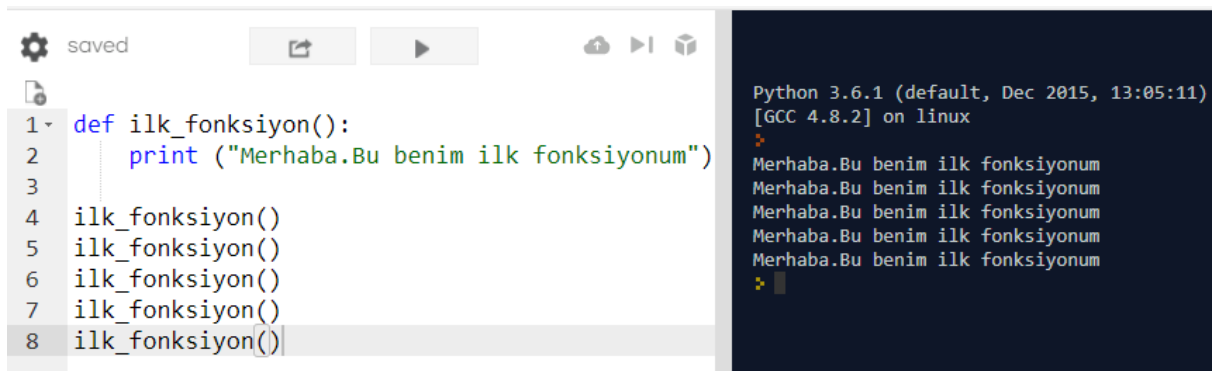
Bir adet TAB yaparak biraz ieriye doęru giriyor ve fonksiyonumuzu yazmaya bařlıyoruz. Burada print(“Merhaba. Bu benim ilk fonksiyonum”) dedik. Yani bu fonksiyon her aęırıldıęında ekrana bu yazı yazacak. Ama henüz aęırmadık. ilk_fonksiyon() ifadesiyle fonksiyonumuzu aęırıyoruz.



```
saved [share] [run] [cloud] [stop] [cube]
1 def ilk_fonksiyon():
2     print ("Merhaba.Bu benim ilk fonksiyonum")
3
4     ilk_fonksiyon()
5
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Merhaba.Bu benim ilk fonksiyonum
```

Görüldüęü gibi ekranda Merhaba. Bu benim ilk fonksiyonum yazdı. Eęer fonksiyonu 5 kez aęırırsak aynı yazı 5 kez yazıyor.

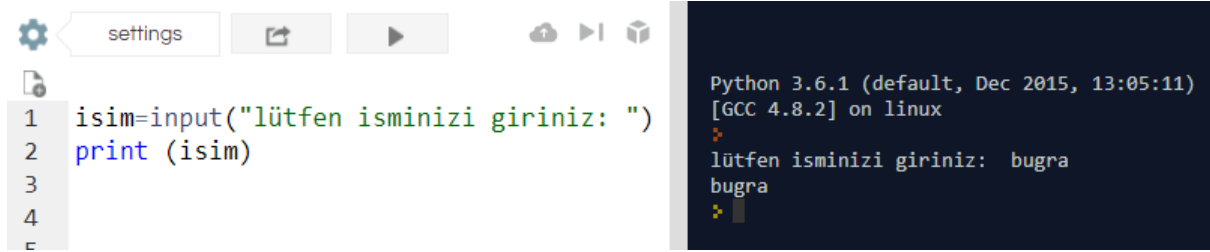


```
saved [share] [run] [cloud] [stop] [cube]
1 def ilk_fonksiyon():
2     print ("Merhaba.Bu benim ilk fonksiyonum")
3
4     ilk_fonksiyon()
5     ilk_fonksiyon()
6     ilk_fonksiyon()
7     ilk_fonksiyon()
8     ilk_fonksiyon()
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
Merhaba.Bu benim ilk fonksiyonum
Merhaba.Bu benim ilk fonksiyonum
Merhaba.Bu benim ilk fonksiyonum
Merhaba.Bu benim ilk fonksiyonum
Merhaba.Bu benim ilk fonksiyonum
```

Peki sadece veri ıkıřı deęil de veri giriři yaptırmak istesek ? Yani kullanıcıdan aldığımız bir veriyi ekranda yazdırsak bunu nasıl yapardık ?

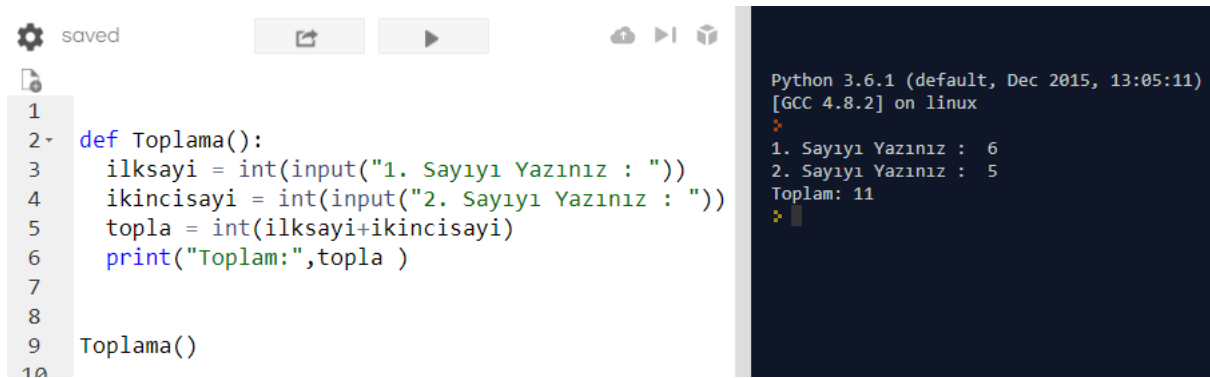
İşte bu noktada da bir başka fonksiyonumuz olan **input** 'u kullanıyoruz.



```
settings [run] [stop] [help]
1 isim=input("lütfen isminizi giriniz: ")
2 print (isim)
3
4
5
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
lütfen isminizi giriniz: bugra
bugra
>
```

Input fonksiyonunu kullanarak kullanıcıdan ismini girmesini istedik. Ardından girilen ismi print fonksiyonu ile ekrana yazdırdık. Şimdi bunu bir adım ileriye götürerek girilen 2 sayıyı toplayan bir fonksiyon yazalım.



```
saved [run] [stop] [help]
1
2 def Toplama():
3     ilksayi = int(input("1. Sayıyı Yazınız : "))
4     ikincisayi = int(input("2. Sayıyı Yazınız : "))
5     topla = int(ilksayi+ikincisayi)
6     print("Toplam:",topla )
7
8
9 Toplama()
10
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
1. Sayıyı Yazınız : 6
2. Sayıyı Yazınız : 5
Toplam: 11
>
```

Bu programda önce Toplama() isimli fonksiyonumuzu oluşturduk. Ardından ilksayi ve ikincisayi yi input fonksiyonu ile kullanıcıdan aldık. Sayıları toplama işlemine sokup print fonksiyonuyla toplamı ekrana yazdırdık. Aslında burada fonksiyon içerisinde başka bir fonksiyonun kullanımını da görmüş olduk. Programımızda parantezlerin dışında yer alan **int** ifadeleri ise kullanıcının girdiği sayıların Integer türünde olduğunu kontrol ediyor. Eğer bir tam sayı yerine 3.5 gibi bir ondalıklı sayı veya "Ahmet" gibi bir kelime girersek program hata verecektir.

Şimdi matematiksel bir fonksiyon yazalım. Yarıçapı girilen bir dairenin alanını hesaplayan uygulama yapalım. Normalde bunu $\pi \cdot r$ fonksiyonuyla yapıyoruz. Girilen yarıçapın karesini alıyoruz ardından bunu matematiksel bir sabit olan π (pi) sayısıyla çarpıyoruz. Fonksiyonumuzu yazalım. Adı alan_hesapla olsun.

```
saved [share] [run] [info] [exit]
1 def alan_hesapla():
2     pi = 3.14
3     r = float(input("Yarıçap giriniz:"))
4     alan = r*r*pi
5     print(alan)
6
7
8 alan_hesapla()
9
```

```
Python 3.6.1 (default, Dec 2015,
[GCC 4.8.2] on linux
>
Yarıçap giriniz: 4
50.24
>
```

Şimdi yukarıdaki fonksiyonumuzda ne yaptığımıza daha yakından bakalım.

Önce alan_hesapla() isimli fonksiyonumuzu oluşturduk. Fakat henüz boş. İçine önce pi=3.14 diyerek bir değişken ekledik. Pi sayısının değerini 3.14 kabul edeceğimizi söylemiş olduk. Ardından r= float(input("Yarıçap giriniz:")) satırıyla kullanıcıdan float değişken türünde bir yarıçap girilmesini istedik. Eğer Integer olarak bu kısmı alsaydık kesirli bir sayı olan 3.14 ile çarparken sorun çıkacaktı. Program hata verecekti. Ardından alan formülümüzü yazdı. r*r*pi ile alan hesapladık. Print fonksiyonuyla bulunan alanı ekrana yazdık.

Evet fonksiyonumuzu yapmıştık fakat henüz çağırmamıştık. alan_hesapla() koduyla fonksiyonumuzu çalıştırdık. Çıktı ekranında bizden yarıçap değeri girmemiz istendi. Biz de 4 girdik. Ardından program otomatik olarak bize dairenin alanını hesapladı. 50.24 buldu. Eğer daha gerçekçi bir sonuç olmasını istiyorsak bu sonucu 50.24 cm veya metre gibi birimiyle beraber ekrana yazdırabiliriz.

Biraz soluklanıp sohbet edelim. Şimdiye kadar olan fonksiyonlarda bir şey dikkatinizi çekt mi ? Düşünün bakalım ne olabilir.

Ben söyleyeyim, şimdiye kadar olan fonksiyonlarda, fonksiyonumuz herhangi bir parametre almıyordu. Yani **alan_hesapla()** şeklinde doğrudan çalıştırıyorduk. Peki buradaki parantezlerin içi hep boş mu kalacak ? Bir değer alamaz mı ? Yani alan_hesapla(4) diyemez miyiz ? Böyle olsa daha güzel olur bence. Hemen 4 ün alanı hesaplanır.

```
saved [run] [play] [stop] [refresh] [close]
1 def alan_hesapla(r):
2     pi = 3.14
3     alan = r*r*pi
4     print(alan)
5
6 alan_hesapla(4)
7 print("Şimdi yarıçapı 5 yapalım")
8 alan_hesapla(5)
9
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
50.24
Şimdi yarıçapı 5 yapalım
78.5
```

Yukarıda görüldüğü gibi bu sorunu pratik bir şekilde çözdük. İlk başta **def alan_hesapla()** yerine **def alan_hesapla(r)** dedik. Yani Python'a dedik ki "Bak sevgili Python, fonksiyonumun içerisinde bir tane r değişkeni var. Ben fonksiyonu çalıştırırken bu r değişkenini sana söyleyeceğim. Sen içerideki kodlarda bunu kullan."

Bunu yaptıktan sonra fonksiyonu artık alan_hesapla() şeklinde değil de alan_hesapla(4) , alan_hesapla(5) gibi ifadelerle çağırdık. 4 için 50.24 5 içinse 78.5 sonucunu bize verdi. İşte bir parametreyle fonksiyon kullanmak bu kadar basit. Python sen ne güzel şeysin !

Hep sayılarla uğraştık bir de sözcükler ile fonksiyon yazalım. Diyelim ki bir robot yaptık , robotumuz istediğimiz kişinin doğum gününü kutluyor. Nasıldı doğum günü kutlaması. "İyi ki doğdun, iyi ki doğdun, iyi ki doğdun Buğra!" gibi. İki kere iyi ki doğdun diyoruz. Ardından üçüncü iyi ki doğdun ile beraber ismi söylüyoruz. Şimdi fonksiyonumuzu yapalım.

```
saved [run] [play] [stop] [refresh] [close]
1 def dogumgunu_kutla(isim):
2     print ("İyi ki doğdun!")
3     print ("İyi ki doğdun!")
4     print ("İyi ki doğdun", isim)
5
6
7 dogumgunu_kutla("Ahmet")
8 dogumgunu_kutla("Mehmet")
9 dogumgunu_kutla("Sevda")
10
```

```
Python 3.6.1 (default, Dec
[GCC 4.8.2] on linux
İyi ki doğdun!
İyi ki doğdun!
İyi ki doğdun Ahmet
İyi ki doğdun!
İyi ki doğdun!
İyi ki doğdun Mehmet
İyi ki doğdun!
İyi ki doğdun!
İyi ki doğdun Sevda
```

Görüldüğü gibi önce Ahmet'in sonra Mehmet'in ve en son da Sevda'nın doğum gününü kutladık. Şimdi kelimeler ve sayıları beraber kullanalım. Fonksiyonumuz bir kaç parametre alsın. Şöyle bir senaryo düşünelim. Bilge isminde çalışkan bir

öğrencimiz var. İlk sınavdan 100 almış fakat ikinci sınavı biraz kötü geçmiş 70 almış. Sisteme bu bilgileri girerek , not ortalamasını hesaplayacağız. Fonksiyonumuz 3 parametre alacak. İlki “Bilge” ikincisi 100 ve üçüncüsü 70 olacak.

```
saved [run] [stop] [refresh] [close]
1 def not_hesapla(isim,not1,not2):
2     print(isim,"isimli öğrencinin notu:")
3     ortalama = int((not1+not2)/2)
4     print("Ortalama:", ortalama)
5
6
7     not_hesapla("Bilge",100,70)
8
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Bilge isimli öğrencinin notu:
Ortalama: 85
>
```

Görüldüğü gibi programımız isim kısmını Bilge isimli öğrencinin notu ifadesi içerisinde kullandı. Notlardan ise bir ortalama olarak Ortalama: ifadesinin karşısında sonucu hesapladı.

Bu örneğimizle fonksiyonlar konusunu tamamladık. Kendi fonksiyonlarınızı yazmak için ne bekliyorsunuz ? :)

Bölüm 10 : Operatörler

Dikkat ettiniz mi bilmiyorum ama günlük hayatta bazı kelimeler işimizi çok kolaylaştırıyor. Mesela şu kelimelerin olmadığını düşünün: “ve, ya da, eşittir, büyüktür, küçüktür, topla, çıkar” Bir anda elimiz ayağımıza karıştırdı. Gerçi son zamanlarda “aynen” diye bir kelime var ama bunu henüz Python keşfetmedi. Python bizi duyuyorsa “aynen” için de özel bir kod istiyoruz :)

Neyse konuyu dağıtmayalım. İşte bu kelimeler günlük hayatta nasıl işimizi kolaylaştırıyorsa programlamada da aynı şekilde işimize yarıyor. Programlama dilinde bu ifadelerin kullanımına operatörler diyoruz. Bu operatörler kendi içlerinde çeşitli bölümlere ayrılıyor.

Kısaca bakarsak Python aşağıdaki operatörlerden oluşuyor.

- Aritmetik (Matematiksel) Operatörler
- Karşılaştırma Operatörler
- Atama Operatörleri
- Mantıksal Operatörler
- Üyelik Operatörleri

- Kimlik Operatörleri

Aritmetik Operatörler

Aritmetik operatörler bizim Matematik işlemlerinde kullandığımız operatörler aslında. Matematik denildiğinde hangi işlemler aklımıza geliyor. "Toplama, çıkarma, çarpma, bölme, kalan bulma" değil mi? Ayrıca ilerleyen sınıflarda üs alma işlemi de görüyoruz. İşte bu işlemlerin aynılarını Python'da operatörler ile yapıyoruz. Şimdi bu operatörleri bir tabloda gösterelim.

Operatör	Fonksiyonu	Örnek İşlem (a= 4 , b= 3)
+	Toplama	$a+b = 7$
-	Çıkarma	$a-b = 1$
*	Çarpma	$a*b = 12$
/	Bölme	$a/b = 1.333$
%	Kalan Bulma	$a\%b = 1$
**	Üs Alma	$a**b = 64$
//	Kalansız Bölme	$a//b = 1$

Karşılaştırma Operatörleri

Karşılaştırma operatörleri adı üstünde olduğu gibi iki değişkeni karşılaştırıyor. Bu bir sayı da olabilir, bir metin de olabilir. Tablomuzda bu operatörlerin nasıl kullanılacağına, fonksiyonlarına bakalım. Bunları örnek bir işlem ile gösterelim.

Operatör	Fonksiyonu	Örnek İşlem (a= 10, b=5)
==	İki değer eşitse "true" sonucunu verir..	$(a == b)$, "false" sonucunu verir.
!=	İki değer eşit değilse "true" sonucunu verir.	$(a != b)$, "true" sonucunu verir.
<>	İki değer eşit değilse "true" sonucunu verir. != operatörü ile aynı işlevi görür.	$a <> b$, "true" sonucunu verir.

>	İlk değer büyükse "true" sonucunu verir.	$a > b$, "true" sonucunu verir.
<	İlk değer büyükse "false" sonucunu verir.	$a < b$, "false" sonucunu verir.
>=	İlk değer büyükse veya eşitse "true" sonucunu verir.	$a \geq b$, "true" sonucunu verir.
<=	İlk değer küçükse veya eşitse "true" sonucunu verir.	$a \leq b$, "false" sonucunu verir.

Atama Operatörleri

Bir yemek yaptığınızı düşünün. Bardaktaki suyu tencereye, kaptaki unu bardağa dökeriz. Bir sürü bunun gibi işlem yaptıktan sonra yemeğimiz hazır olur. İşte kodlama yaparken de aslında sürekli bir şeyi diğer yere taşırız. En basit bir toplama işleminde bile toplam = a + b derken a ve b değişkenlerindeki değerleri toplam isimli değişkene atarız. Tıpkı bir su ve unu bir tencereye dökmek gibi. İşte bu işlemler sırasında yaptığımız şeye "atama" diyoruz. Atama işlemlerini de "Atama Operatörü" adı verilen semboller ile yapıyoruz. Kafanız karışmasın, tabloyu görünce çok daha iyi anlayacaksınız.

Operatör	Fonksiyonu	Örnek İşlem (a= 5, b = 4)
=	Sol tarafta istenen işlemin sonucunu sağ tarafa atar.	$a + b = 9$ $a - b = 1$
+=	Sağdaki değeri, soldaki değerle toplar ve soldaki değere yazar.	$a += b$ şu anlama gelir. $a + b = a$
-=	Sağdaki değeri, soldaki değerden çıkarır ve soldaki değere yazar.	$a -= b$ şu anlama gelir. $a - b = a$
*=	Sağdaki değeri, soldaki değerle çarpar ve soldaki değere yazar.	$a *= b$ şu anlama gelir. $a * b = a$
/=	Sağdaki değeri, soldaki değere böler ve soldaki değere yazar.	$a /= b$ şu anlama gelir. $a / b = a$
%=	Sağdaki değer soldaki değerden kalanını hesaplar ve soldaki değere atar.	$a \% b$ şu anlama gelir. $a \% b = a$
**=	Sağdaki değeri, soldaki değere üs olarak alır , soldaki değere atar.	$a ** b$ şu anlama gelir. $a ** b = a$
//=	Sağdaki değeri, soldaki değere kalansız olarak böler ve sonucu soldaki değere atar.	$a // b$ şu anlama gelir. $a // b = a$

-“Buğra abi bir şey soracağım.”

-”Efendim?”

-”Burada çok saçma bir şey var. Mesela $a+=b$, $a+b=a$ anlamına geliyorsa neden ikincisini kullanmıyoruz, kafamızı karıştırıyoruz.”

-”Eğer istersen ikincisini de kullanabilirsin. Bu bir tür kısaltma sağlıyor. Aslında ilk duyduğumda ben de çok gıcık olmuştum. Ama şöyle düşün. Bir manav programı yazıyorsun. Bu manava sürekli yeni karpuzlar geliyor. Gelen karpuzları sayacağın zaman her seferinde $olankarpuzsayısı = olankarpuzsayısı + gelenkarpuzsayısı$ diye düşünmektense $olankarpuzsayısı+=gelenkarpuzsayısı$ demen işini kolaylaştırır. Aynısı diğer işlemlerde de geçerli. Programlar yazdıkça bu kısaltmanın kolaylığının farkına varacağına eminim.

Mantıksal Operatörler

Başta konuşmuştuk. Bazı kelimeler işimizi kolaylaştırır. Diyelim ki bir elbise satın alacaksınız. Satıcıya şöyle diyebilirsiniz. “Yeşil olsun ve orta beden olsun.” Bu sayede istediğiniz elbiseyi daha rahat bulabilirsiniz. Satıcı size “Orta boy elbisemiz var fakat yeşil renkte kalmadı. Başka bir renk olabilir mi?” dedi ve siz de “Kırmızı veya mavi olabilir.” dediniz diye düşünelim. İşte bu ve, veya kelimelerinin programlamadaki karşılıklarına “Mantıksal Operatörler” diyoruz. Hemen tablomuzdan örneklerle inceleyelim.

Operatör	Fonksiyon	Örnek İşlem
and	İki durumda doğruysa True sonucunu verir.	(matematik = “gecti” and turkce=”gecti”) ise sınıfı geçme durumuna True cevabını verebiliriz.
or	İki durumdan en az biri doğruysa True sonucunu verir.	(anne = “Türk” or baba=”İtalyan”) ise Türkiye Cumhuriyeti Vatandaşı olma durumu sonucu True cevabını verir. Çünkü Türkiye Cumhuriyeti vatandaşı olmak için anne veya babadan birinin Türk olması yeterlidir.
not	And ve or operatörlerinin başına geldiğinde True yerine False, False yerine True şeklinde sonuç oluşmasını sağlar.	not((matematik = “gecti” and turkce=”gecti”)) ifadesiyle And örneğimizi düşünelim. Sınıfı geçme durumumuz False olacaktı. Aynı şekilde T.C vatandaşı olma sonucumuz da false olacaktı.

Üyelik Operatörü

Python’da değişken tiplerini incelerken listeler, demetler, sözlüklere değinmiştik. Bu değişkenlerin içerisinde birden çok eleman olabiliyordu. Biz de bir elemanın bu değişkenin içerisinde olup olmadığını **in** operatörü ile kontrol edebiliyorduk. Aynı şekilde **in** operatörünün tam tersi durum için de **not in** operatörünü kullanabiliyoruz. İşte bu iki operatör bizim Python’daki mantıksal operatörlerimizi oluşturuyor. Tablomuzdan bir örnekle inceleyelim.

Operatör	Fonksiyon	Örnek İşlem (a=10, liste= [10,20,30]
in	“İçinde mi?” sorusunu sorar. İçindeyse True cevabını verir.	a in liste kodu True sonucunu verir.
not in	“İçinde değil mi?” sorusunu sorar. İçinde değilse True cevabını verir.	a not in liste kodu ise False sonucunu verir.

Kimlik Operatörleri

Kimlik operatörleri “Eşit mi?” sorusunu sorar. İki tane kimlik operatörümüz var. İlki “is” ikincisi ise “is not” olarak kullanılıyor. Eğer iki değer eşitse is operatörü True, değilse False sonucunu veriyor. İkinci operatörümüz olan “is not” ise bunun tam tersini yapıyor.

Operatör	Fonksiyon	Örnek İşlem (a=10, b=20)
is	“Eşit mi?” sorusunu sorar. Eşitse True cevabını verir.	a is b kodu False sonucunu verir.
is not	“Eşit değil mi?” sorusunu sorar. Eşit değilse True cevabını verir.	a is not b kodu ise True sonucunu verir.

Bölüm 11 : Koşullu Durumlar

Isırmayan yılan Python’da değişkenler, fonksiyonlar derken baya bir yol aldık. Fakat eksik bir şey var. Şimdiye kadarki örneklerimizde hep bir olasılık üzerinden gittik. Yani eğer şu olursa ekrana bunu yazdır gibi. Fakat gerçek hayat böyle değil. Bir kahve makinesine para atıp içecek alacağımızı düşünelim. Makina 11 numarasını girersek normal kahve 12 numarasını girersek sütlü kahve ve 13 numarasını girersek sıcak çikolata veriyor. İşte burada karşılaştığımız duruma “Koşullu Durum” ismini veriyoruz. Bir durum ifadesi kullanmadığımız takdirde, önceki bölümlerde öğrendiğimiz kodlarla, bunu yapmamız ne yazık ki mümkün değil.

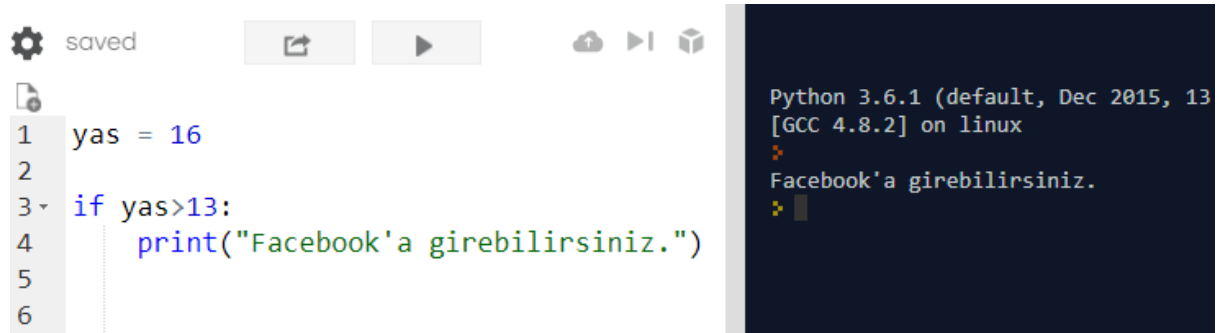
Programımız şöyle çalışıyor.

Eğer 11 numarayı girerse,

normal kahve ver
Eğer 12 numarayı girerse,
sütlü kahve ver
Eğer 13 numarayı girerse
sıcak çikolata ver.

İşte bu çalışmayı kodlarla yapacağız. Koşullu ifadelerde kullandığımız deyimler var. Bunlar if, elif ve else şeklinde. Elif aslında else if in kısaltması, sizin arkadaşınız olan Elif ile alakası yok :)

Öncelikle if deyimini ile başlayalım. If İngilizce’de eğer anlamına geliyor ve neredeyse bütün programlama dillerinde de aynı amaçla kullanılıyor. Günlük hayattan bir örnekle başlayalım. Yasalara göre 13 yaşından küçük biri Facebook, Twitter gibi sosyal ağlara giremez. Siteye kayıt olurken ziyaretçinin 13 yaşından büyük olup olmadığı kontrol edilir. Örnek kodumuzda da önce yas=16 diyerek bir yaş girdik. Ardından if yas>13 diyerek eğer yaş 13’den büyükse dedik ve ekrana “Facebook’a girebilirsiniz”) yazdırdık.



```
saved [share] [run] [info] [play] [close]
```

```
1 yas = 16
2
3 if yas>13:
4     print("Facebook'a girebilirsiniz.")
5
6
```

```
Python 3.6.1 (default, Dec 2015, 13:
[GCC 4.8.2] on linux
Facebook'a girebilirsiniz.
```

Burada çok önemli bir nokta var. Dikkat edersek print ile başlayan satır biraz içeriden başlıyor. If ile başlayan satırın altında da küçük noktalar halinde aşağıya inen bir çizgi var. Python parantez açıp kapatmakla uğraşmamız için bir kolaylık yapmış. Diyor ki if ile başlayan koşul gerçekleştiğinde neyin gerçekleşmesini istiyorsan bunu bir sekme içeriden yaz. Ben anlarım. Şimdi bu anlattığımızı bir örnekle gösterelim. Print ile başlayan 3 satırımız da aynı miktarda içeriye girmiş ve soruya 16 şeklinde cevap verince Python 3 cümleyi birden ekrana yazdırıyor.



```
settings [share] [run] [info] [play] [close]
```

```
1 yas = 16
2
3 if yas>13:
4     print("Facebook'a girebilirsiniz.")
5     print("Demek 18 yaşına 2 yıl kaldı.")
6     print("Nasıl gidiyor okul ?")
7
8
```

```
Python 3.6.1 (default, Dec 2015, 13:
[GCC 4.8.2] on linux
Facebook'a girebilirsiniz.
Demek 18 yaşına 2 yıl kaldı.
Nasıl gidiyor okul ?
```

Peki yaşı biz doğrudan 16 olarak yazmasaydık, kullanıcı girseydi nasıl olurdu ? Input fonksiyonu ile bunu yapalım.

```
saved [share] [run] [upload] [refresh] [close]
1 yas = int(input("Yaşınızı girin:"))
2
3 if yas>13:
4     print("Facebook'a girebilirsiniz.")
5
6
```

```
Python 3.6.1 (default, Dec 2015,
[GCC 4.8.2] on linux
>
Yaşınızı girin: 17
Facebook'a girebilirsiniz.
>
```

Yukarıdaki kodumuzda yas değişkenini input fonksiyonu ile aldık. Alırken de dışarıda bir int ifadesi kullanılarak girilen içeriğin Integer yani tam sayı olarak algılanmasını sağladık.

Günlük hayattan bir başka durumla devam edelim. İnternet sitelerinde sürekli olarak şifre giriyoruz değil mi ? Şimdi kendi kendimize bir şifre giriş sistemi yapalım.

```
saved [share] [run] [upload] [refresh] [close]
1 print("Gmail'e hoşgeldiniz.")
2 kullanıcıadı = input("Kullanıcı adı girin:")
3 sifre = int(input("Şifre girin:"))
4
5 if kullanıcıadı=="bugraayan" and sifre== 131313:
6     print("Hesaba giriş yapıldı.")
7
```

```
Python 3.6.1 (default, Dec 2015,
[GCC 4.8.2] on linux
>
Gmail'e hoşgeldiniz.
Kullanıcı adı girin: bugraayan
Şifre girin: 131313
Hesaba giriş yapıldı.
>
```

Örneğimizde sanki Gmail'e bağlanıyormuş gibi bir küçük program yaptık. Önce kullanıcı adımız ve sifremiz input girişi ile istendi. Ardından koşullu durum kısmında "Eğer kullanıcı adı bugraayan ifadesine ve şifre 131313'e eşitse giriş yap dedik. Derleyici çalıştığında sırasıyla bizden kullanıcı adı ve şifre istendi. Doğru değerler girildiğinde de ekranda "Hesaba giriş yapıldı" yazısı geldi.

Peki hatalı giriş yapsaydık ? Yani başka bir ihtimal olsaydı ne olacaktı ? Hadi bunun üzerine biraz düşünelim. "Elma dersem çık, armut dersem çıkma" diye küçükken oynadığımız bir oyun vardı. Bu oyunu bir kod haline getirelim.

```
saved share run
1 print("Hadi senle bir oyun oynayalım.")
2 meyve = input("Bir meyve söyler misin:")
3
4 if meyve == "elma":
5     print("Çık")
6 elif meyve=="armut":
7     print("Çıkma")
8
```

```
Python 3.6.1 (default, Dec 2015,
[GCC 4.8.2] on linux
>
Hadi senle bir oyun oynayalım.
Bir meyve söyler misin: elma
Çık
>
```

İşte ünlü elif ile tanıştık. Elif aslında “Else ve if” deyimlerinin kısaltmasından oluşuyor. If ile eğer şu ihtimal olursa diye söze başlıyoruz , elif ile ise başka bir ihtimalden bahsediyoruz. Sadece bir tane if kullanıyoruz fakat dilediğimiz kadar elif kullanabiliriz. Örneğin mevcut kodumuza Portakal ve Şetali olasılıklarını da ekleyelim.

```
saved share run
1 print("Hadi senle bir oyun oynayalım.")
2 meyve = input("Bir meyve söyler misin:")
3
4 if meyve == "elma":
5     print("Çık")
6 elif meyve=="armut":
7     print("Çıkma")
8 elif meyve == "portakal":
9     print("Portakal nereden çıktı şimdi")
10 elif meyve == "şeftali":
11     print("Şeftali severim aslında.")
12
```

```
Python 3.6.1 (default, Dec 2015, 1
[GCC 4.8.2] on linux
>
Hadi senle bir oyun oynayalım.
Bir meyve söyler misin: portakal
Portakal nereden çıktı şimdi
>
```

Portakal yazınca program bize “Portakal nereden çıktı şimdi?” cevabını verdi. Cidden portakal nereden çıktı şimdi. :)

Peki bütün meyveleri böyle yazabilir miyiz? Çok zor değil mi ? İşte tam da bunun için else ifadesini kullanıyoruz. Else geri kalan tüm ihtimalleri kapsıyor. Hemen ekleyelim.

```
saved share run
1 print("Hadi senle bir oyun oynayalım.")
2 meyve = input("Bir meyve söyler misin:")
3
4 if meyve == "elma":
5     print("Çık")
6 elif meyve=="armut":
7     print("Çıkma")
8 elif meyve == "portakal":
9     print("Portakal nereden çıktı şimdi")
10 elif meyve == "şeftali":
11     print("Şeftali severim aslında.")
12 else:
13     print("Bu meyve ile ilgili bilgin yok.")
14
```

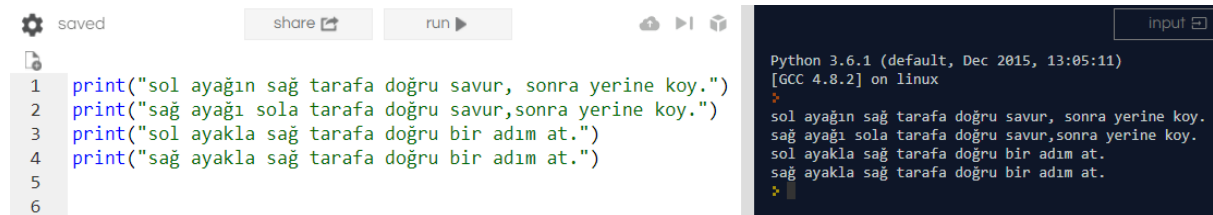
```
Python 3.6.1 (default, Dec 2015,
[GCC 4.8.2] on linux
>
Hadi senle bir oyun oynayalım.
Bir meyve söyler misin: muşmula
Bu meyve ile ilgili bilgin yok.
>
```

Yukarıda muşmula yazdık ve “Bu meyve ile ilgili bilgim yok.” cevabını aldık. Aynı şekilde elma,armut,portakal ve şeftali dışında herhangi bir değer girdiğimizde de aynı cevabı alacaktık. Else kullanımıyla bu konuyu tamamladık. If, elif ve else kullanımının olduğu satırdaki iki nokta : işaretine dikkat edelim. Eğer bu iki nokta işaretlerini koymazsak programımız çalışmayacaktır.

Bölüm 12 : Döngüler

Döngü denildiğinde aklınıza ne geliyor? Şöyle gözlerinizi kapatıp düşününce benim aklıma “Dönme Dolap” geliyor. Kimine ise uykusunu getirmek için atlayan koyunları saymak gelir. Uykumuzu getirecek kadar sıkıcı bir şey mi bu döngüler ? Koyunları bilmem ama ısırmayan yılanımız Python’da döngüler son derece eğlenceli. Hatta kanka olacağınızı düşünüyorum.

- Tamam da Buğra abi nerede kullanacağız döngüleri ?
- Düğünde oynarken kullanacağız tabi ki.
- Dalga geçme abi ya ne düşünüy.
- Tamam hemen anlatıyorum. Mesela bir akrabanın düşününe gittin. Bir baktın etrafta hareketlilik başladı. Hiç oynamayı sevmemene rağmen sana doğru bakışları farkettiler. Tehlike yaklaşıyor biri gelip seni halaya kaldıracak. Kolundan çekmek için yaklaşıyorlar. Artık neyse, oynayayım bari dedin. Çıktın ve “Nasıl oynayacağız?” dedin. İşin uzmanı bir amca gelip dedi ki “Halay oyunu sol ayağın sağ tarafa doğru savrulup tekrar yerine koyulması ile başlar. Sonra aynı şekilde sağ ayak sola doğru savrulup yerine koyulur ve sırasıyla önce sol ayakla, sonra da sağ ayakla sağ tarafa doğru birer adım atılır.” Sen iyi bir programcısın. Hemen kafanda bunu yazılıma döktün.



```
1 print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
2 print("sağ ayağı sola tarafa doğru savur, sonra yerine koy.")
3 print("sol ayakla sağ tarafa doğru bir adım at.")
4 print("sağ ayakla sağ tarafa doğru bir adım at.")
5
6
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
sol ayağın sağ tarafa doğru savur, sonra yerine koy.
sağ ayağı sola tarafa doğru savur, sonra yerine koy.
sol ayakla sağ tarafa doğru bir adım at.
sağ ayakla sağ tarafa doğru bir adım at.
```

Tamam güzel. “Bunu kaç kere yapacağız?” “Ne! 100 kere mi ?” Hadi şimdi bunu koda dök bakalım.


```

print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
print("sağ ayağı sola tarafa doğru savur,sonra yerine koy.")
print("sol ayakla sağ tarafa doğru bir adım at.")
print("sağ ayakla sağ tarafa doğru bir adım at.")

print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
print("sağ ayağı sola tarafa doğru savur,sonra yerine koy.")
print("sol ayakla sağ tarafa doğru bir adım at.")
print("sağ ayakla sağ tarafa doğru bir adım at.")

print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
print("sağ ayağı sola tarafa doğru savur,sonra yerine koy.")
print("sol ayakla sağ tarafa doğru bir adım at.")
print("sağ ayakla sağ tarafa doğru bir adım at.")

print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
print("sağ ayağı sola tarafa doğru savur,sonra yerine koy.")
print("sol ayakla sağ tarafa doğru bir adım at.")
print("sağ ayakla sağ tarafa doğru bir adım at.")

```

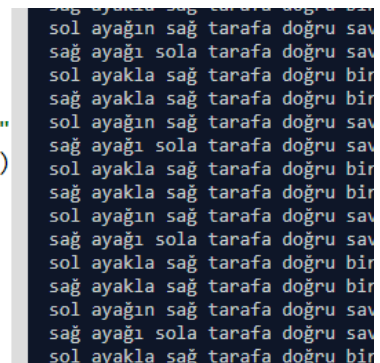
Böyle aşağıya doğru sayfalarca gidiyor kod. Acil bir şey yapman lazım. Bu işi bir döngüye sokmamız lazım. Ne dedik “döngü” mü ? Evet işte doğru kelime. Döngüleri kullanacağız. Burada kullanacağımız deyimimin ismi **for** .

```
for x in range(1,100):
```

```

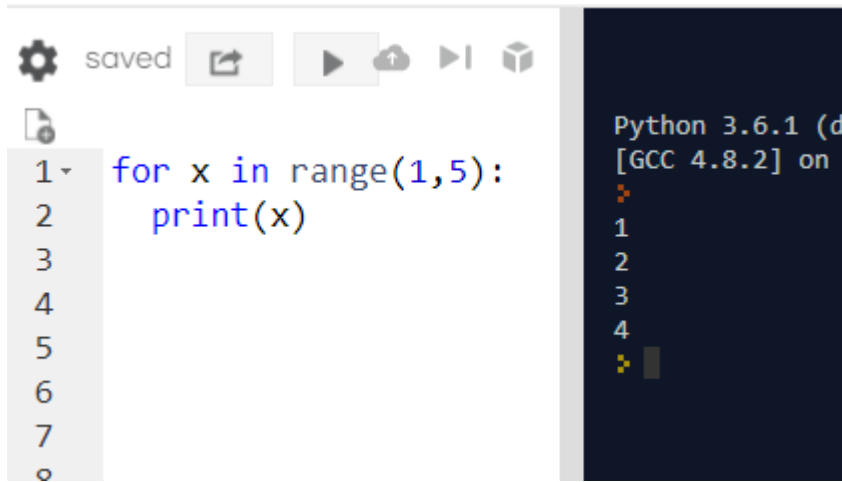
    print("sol ayağın sağ tarafa doğru savur, sonra yerine koy.")
    print("sağ ayağı sola tarafa doğru savur,sonra yerine koy.")
    print("sol ayakla sağ tarafa doğru bir adım at.")
    print("sağ ayakla sağ tarafa doğru bir adım at.")

```



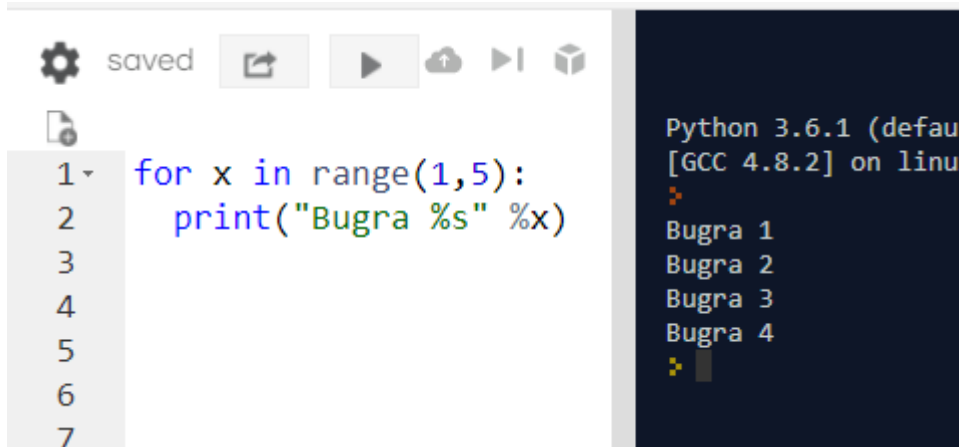
Ekrana yazdırılmasını istediğimiz yazının üstüne **for x in range(1,100):** ifadesini ekledik. Yani; “1’den başlayarak 100’e kadar say. Her seferinde ekrana içerideki kodları yazdır.” dedik. Sonuçta da tam 100 kere aynı satırlar ekrana yazıldı. range fonksiyonunu döngüde sayıları kullandığımızda çalıştırıyoruz. Her döngüde range ifadesini kullanmamıza gerek yok. İlerleyen örneklerde bunu göreceğiz.

Harika ! Devam edelim. Ekrana aynı şeyi değil de içerisinde değişken olan bir şeyler yazdıralım.



```
saved [run] [upload] [stop] [cube]
Python 3.6.1 (d
[GCC 4.8.2] on
1
2
3
4
5
6
7
8
```

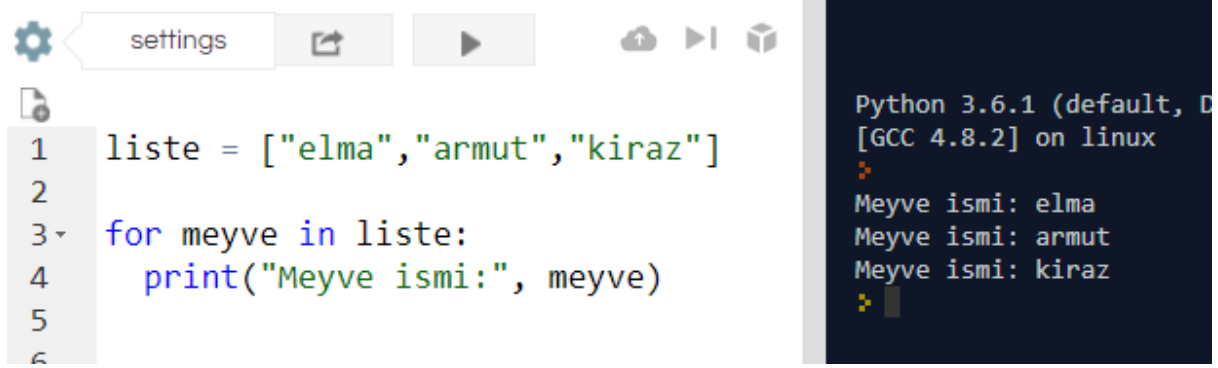
Yukarıdaki kodumuzda aslında şunu yaptık. x'e 1 den 5 e kadar değer ver. Sonra sırasıyla ekrana bu x değerini yazdır. Şimdi sabit bir ifadeyle değişkenimizi birleştirelim. Buradaki %s, %x kısmına sonra değineceğiz. Şimdilik döngüye dikkat edelim.



```
saved [run] [upload] [stop] [cube]
Python 3.6.1 (defau
[GCC 4.8.2] on linux
Bugra 1
Bugra 2
Bugra 3
Bugra 4
5
6
7
```

Güzel gidiyoruz. Peki her zaman range yazıp bir sayı listesini mi ekrana yazacağız. Elbette değil. Kendi listemizi de ekrana yazdırabiliriz. Liste mi dedik? Aaa böyle bir konu işlemiştik değil mi ? Hemen hatırlayalım. Örnek bir liste nasıl oluyordu ?

liste = ["elma", "armut", "kiraz"] şeklinde oluşturabiliyorduk. Köşeli parantezler ile listeyip başlatıp bitiriyor ve elemanları virgül ile ayırıyorduk. Şimdi listemiz ile for döngüsünü birleştirelim.



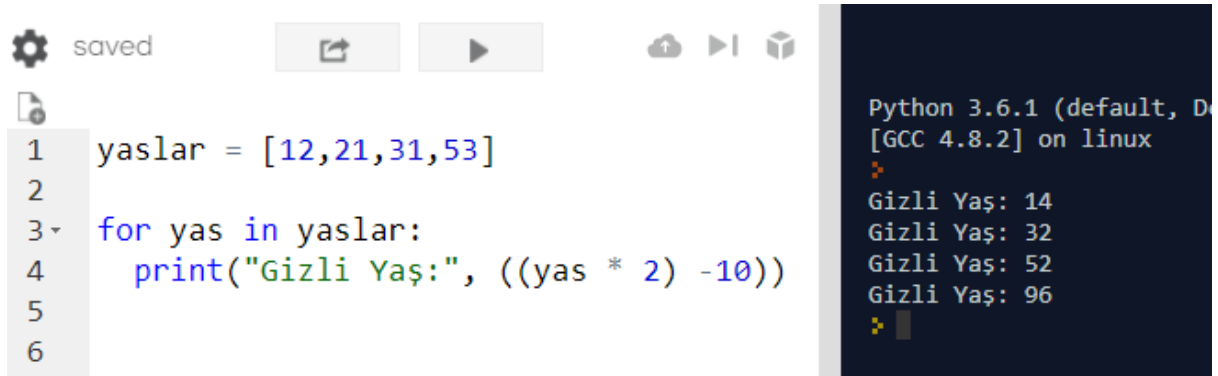
```
settings [run] [stop] [cloud] [play] [cube]
1 liste = ["elma","armut","kiraz"]
2
3 for meyve in liste:
4     print("Meyve ismi:", meyve)
5
6
```

```
Python 3.6.1 (default, D
[GCC 4.8.2] on linux
Meyve ismi: elma
Meyve ismi: armut
Meyve ismi: kiraz
```

Şimdi yukarıdaki örneğimiz inceleyelim. Bir liste tanımladık ve içerisine meyveleri koyduk. Ardından listedeki her meyveyi sırayla ekrana yazdırdık. Yazdırmadan önce de Meyve ismi: şeklinde bir açıklama yazdık.

Döngüler bizim için çok ama çok önemli. O yüzden üzerinde kafa yormaya devam edelim. Listeler ve döngüler ile başka ne yapabiliriz. Örneğin şöyle bir problemimiz olduğunu düşünelim. Gizli yaş diye bir şey hesaplayacağız. Bize yaşını söyleyen kişinin yaşını 2 ile çarpıp 10 çıkarınca onun gizli yaşını buluyoruz. Bunu bir kişide yaptığımızda kolay fakat 10 kişi birden söylediğinde nasıl yaparız ?

İşte programımız bu noktada yardımcı olacak. Kişilerin yaşlarını bir liste olarak alıp tamamının gizli yaşlarını bir kerede hesaplayacağız.



```
saved [run] [stop] [cloud] [play] [cube]
1 yaslar = [12,21,31,53]
2
3 for yas in yaslar:
4     print("Gizli Yaş:", ((yas * 2) -10))
5
6
```

```
Python 3.6.1 (default, D
[GCC 4.8.2] on linux
Gizli Yaş: 14
Gizli Yaş: 32
Gizli Yaş: 52
Gizli Yaş: 96
```

Yaşlarımızı hesapladık. Adım adım gitmeye devam edelim. Şimdi for döngüsü ile if koşul ifadesini birleştirerek bir program yazalım.

Elimizde uzun bir sayı listesi var. Bu sayıların tek mi çift mi olduğunu öğrenmek istiyoruz. Programımız sayıları ikiye bölüyor. Eğer kalan 0 olursa çift, olmazsa tek diye ekrana bilgi veriyor. Hadi deneyelim !

```
saved [run] [play] [share] [info] [close]
1 sayılar = [5,7,13,24,27,30]
2
3 for sayı in sayılar:
4     if sayı%2==0:
5         print(sayı, "sayısı çifttir.")
6     else:
7         print(sayı, "sayısı tektir.")
8
9
```

```
Python 3.6.1 (default, Dec 2015, 1
[GCC 4.8.2] on linux
>
5 sayısı tektir.
7 sayısı tektir.
13 sayısı tektir.
24 sayısı çifttir.
27 sayısı tektir.
30 sayısı çifttir.
>
```

Gayet basit bir kodla bu programı çalıştırdık. Burada `sayı%2` demek sayının ikiye bölümünden kalan anlamına geliyor `sayı%3` deseydik sayının üçe bölümünden kalanı hesaplayacaktı.

Biraz daha işi büyütelim. Fonksiyonları hatırlıyor muyuz ? İşlerimizi daha kolay hale getirmek için kullanıyorduk. Şimdi bir tek-çift fonksiyonu yapalım.

```
saved [run] [play] [share] [info] [close]
1 def tekçift(*sayılar):
2
3     for sayı in sayılar:
4         if sayı%2==0:
5             print(sayı, "sayısı çifttir.")
6         else:
7             print(sayı, "sayısı tektir.")
8
9
10 tekçift(2,5,6,72,71,85)
11
```

```
Python 3.6.1 (default, Dec 2015, 1
[GCC 4.8.2] on linux
>
2 sayısı çifttir.
5 sayısı tektir.
6 sayısı çifttir.
72 sayısı çifttir.
71 sayısı tektir.
85 sayısı tektir.
>
```

`def` ifadesi ile fonksiyonumuzu tanımladık. Fonksiyonumuz bir değişken olarak tek bir değer değil bir kaç değer alacağı için `sayılar` ifadesinin başına bir `*` yıldız koyduk. Ardından fonksiyonumuzun içine önceden çalıştırdığımız kodu yazdık. Burada sekmelere dikkat ediyoruz. Fonksiyonun içinde yer alan kısım dikkat edersek daha içeriden başlıyor.

Son olarak `tekçift(2,5,6,72,71,85)` diyerek elemanların hepsini fonksiyonumuzda işleme soktuk.

Bu örnekte bir fonksiyon, `for` döngüsü ve `if` koşulunu aynı örnekte kullandık. Kendinizi alkışlayabilirsiniz. :)

For döngüsü ile fazlasıyla uğraştık. Şimdi sırada bir başka döngümüz while var. While İngilizce’de “iken” anlamına geliyor. Bunu bizim “koşarken, çalışırken, düşerken” gibi durumlarda geçen iken ifadesi gibi düşünelim. Kodlamada da şöyle diyeceğiz. Bu durum geçerliyken şunu yap. Örneğin; “Kişi yaşını 18 den küçük girerse onu siteye sokma.”

İlk örneğimizde kullanıcı bir sayı giriyor. Programımız da ona Girdiğiniz sayı: diyerek bu sayıyı söylüyor. Ama bunu sonsuz bir döngüyle yapıyor. Hemen çalıştıralım.

```
saved [share] [stop] [run] [refresh] [close]

1 durum = 1
2 while durum == 1 :
3     sayı = input("Bir sayı girin :")
4     print ("Girdiğiniz sayı: ", sayı)
5
6
7
8
```

```
Python 3.6.1 (default, Dec 2015)
[GCC 4.8.2] on linux
>
Bir sayı girin : 3
Girdiğiniz sayı: 3
Bir sayı girin : 5
Girdiğiniz sayı: 5
Bir sayı girin : 4
Girdiğiniz sayı: 4
Bir sayı girin : 
```

Yukarıdaki örnekte ilk olarak durum=1 diyerek bir değişken tanımladık. elma=1 de diyebilirdik veya elma = “yeşil” de diyebilirdik. Burada tanımladığımız durumu aşağıda while ifadesinin karşısına yazdık. Yani dedik ki “Eğer durum 1’e eşitse, aşağıdaki kodları çalıştır.” Kodlarda ise önce input ile bir sayı değişkeni aldık, ardından da bu değişkenimizi ekrana yazdırdık. Durum her zaman 1 olarak kaldığı için program sonsuza kadar bizden sayı girmemizi isteyecek. Görüldüğü gibi 3,5,4 sayılarını sırayla girdik ama program sayı istemeye devam etti.

While döngüsü basit olarak bu şekilde çalışıyor. İstenen durum olduğu sürece belirtilen kodu çalıştırılıyor. Kodun durması için gerekli koşulu da yine biz belirliyoruz.

Diyelim ki ağlayan bir bebek var. Ağlamaya başladığında ona inek taklidi yaptığınızda yani “mööö” dediğinizde susuyor. Başka türlü susturamıyorsunuz. Hadi bunu kodumuzla yazalım.

```
saved [share] [stop] [run] [refresh] [close]

1 konusma = "Sus yavrum."
2
3 while (konusma != "möööö"):
4     konusma = input("Bebek ağlıyor. Bir şey söyle:")
5     print("Ingaaaaaaaaaaaaaaaaaaaa")
6
7 else:
8     print("Bebeği susturdun, tebrikler.")
9
10
11
```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Bebek ağlıyor. Bir şey söyle: sus cocugum
Ingaaaaaaaaaaaaaaaaaaaa
Bebek ağlıyor. Bir şey söyle: aglama cocugum
Ingaaaaaaaaaaaaaaaaaaaa
Bebek ağlıyor. Bir şey söyle: offf sus lutfen
Ingaaaaaaaaaaaaaaaaaaaa
Bebek ağlıyor. Bir şey söyle: möööö
Ingaaaaaaaaaaaaaaaaaaaa
Bebeği susturdun, tebrikler.
>
```

Şimdi programımızı inceleyelim. İlk olarak konuşma kısmını “Sus yavrum” olarak başlattık. Fakat bebek böyle diyince susmayacak. while döngümüzde dedik ki eğer konuşma!= “mööö” yani konuşma “mööö” ye eşit olmazsa bebek “Ingaaaaa” diyerek ağlasın. Else ile eğer bu olmazsa yani konuşma “mööö” ye eşit olursa bebek ağlamayı durdursun. Kodumuz sorunsuz çalıştı. Eğer mööö demeseydik bebek sabaha kadar ağlamaya devam edecekti. :) Bu arada else ifadesi if döngüsünde olduğu gibi bir işe yarıyor. Diğer ihtimaller için olabilecekleri söylüyor, unutmayalım.

Bölüm 13: Modüller

Bu bölümde modülleri anlamaya çalışacağız. Ama önce bir şeyler konuşmamız lazım. Diyelim ki tatilde akrabana ziyarete gideceksiniz. Akrabanız sıcak bir yerde yaşıyor. Denize de girme şansınız olabilir. Evden çıkmadan önce ne yaparsınız ? Bütün giysilerinizi, ayakkabılarınızı, şemsiyenizi, çantanızı alırsanız dışarı çıkabilir misiniz ? Bunlar ağırlık yapar ve evden çıkamazsınız değil mi ? Bunun yerine dolabı açıp yazın giymeye daha uygun giysilerinizi seçersiniz. Mesela kazakları yanınıza almanıza gerek yok. Kışlık eldivenleri de evde bırakabilirsiniz. Fakat yazlık terliklerinizi, kısa kollu t-shirtlerinizi alabilirsiniz.

İşte Python’da kod yazarken biz de öyle yapıyoruz. Modül dediğimiz şeyler sizin programı yazarken yanınıza alacağınız şeyler. Bunları import ifadesi ile programımıza ekliyoruz.

-“Buğra abi niye bizi uğraştırıyorlar. Hepsi programda yüklü olsun, istediğimizi çalıştıralım. import ile eklememize ne gerek var?”

-“Ama bütün modüller yüklü olursa program çok büyük ve yavaş olur. Bunu ister misin?”

Bunu kimse istemez. İşte bu yüzden tıpkı sıcak havada dışarı çıkarken güneş gözlüğünü yanımıza aldığımız gibi matematik ilgili bir program yazarken matematik modülünü, grafiklerle ilgili bir program yazarken grafik modülünü kodumuzun içine ekliyoruz. Bu sayede tek satır kodla programımıza bir çok yeteneği kazandırmış oluyoruz.

Hadi test edelim.

İlk önce matematik.py isminde bir dosya oluşturduk. Py uzantısı Python’ın kısaltması. Tıpkı müzik dosyalarında mp3 olduğu gibi burada da py uzantısını kullanıyoruz.

```
< > main.py matematik.py
1 def toplama( sayi1,sayi2 ):
2
3     print ("Toplam:",sayi1+sayi2)
4
5
6
```

İki satırdan oluşan bir kodumuz var. İlk satırda toplama isminde bir fonksiyon oluşturduk. Bu fonksiyonun çalışması için sayi1 ve sayi2 isminde iki tane değişken gerekiyor. İkinci satırda da print ifadesiyle bu iki değişkenin toplamını ekrana yazdırdık. Artık matematik modülümüz hazır. Dilediğimiz dosyadan **import matematik** diyerek bu dosyayı ve içindeki fonksiyonları çağırabiliriz.

Şimdi main.py dosyamıza geçelim. Python programlarında main.py dosyası bizim ana dosyamız olur. Main İngilizce'de ana anlamına geliyor.

```
< > main.py matematik.py
1 import matematik
2
3 matematik.toplama(4,5)
4
5
```

Ekran çıktısı: **Toplam: 9**

Yine iki satırdan oluşan bir kodumuz var. İlk satırda import matematik diyerek matematik modülünü kullanılabilir hale getiriyoruz. Ardından matematik.toplama(4,5) diyerek 4 ve 5 sayılarını topluyoruz. Ekran'da Toplam: 9 yazıyor. Biz matematik modülünün içerisine 100lerce matematik işlemi de ekleyebilirdik. Örneğin; karekök al, karesini bul, küpünü bul vs. Yine sadece import matematik diyerek bu fonksiyonların tamamını ana fonksiyonumuzda çalışır hale getirebilecektik. Biz aslında bir modülü, modül olduğuna değinmeden kullanmıştık. Kim hatırlıyor ? Matematik modülü değil mi ? import math diyerek matematik modülünü eklemiş ve ardından bir çok matematiksel işlem yapmıştık.

Şimdi ise detaylı olarak inceleyeceğiz. Modüller genel olarak ikiye ayrılıyor. İlki Python'ın içerisinde gelen modüller, ikincisi ise başka programcıların geliştirdiği kullanabileceğimiz modüller.

-”Başka programcılar mı ? Onlar kim Buğra abi ?”

-”Şimdi biz ne yapıyoruz ? Python kodlama dilini öğreniyoruz. Bizimle beraber dünyanın bir çok yerinde insanlar öğreniyor. Bir çok da bilen insan var. Mesela diyelim ki Ahmet diye birisi Twitter ile ilgili işlemler yapan bir program geliştirdi. Aynı şekilde yüzlerce kişi daha buna ihtiyaç duyacak. Ahmet internete Twitter modülünü koyuyor ve diğer insanlar da bunu kullanabiliyor. Harika bir şey paylaşmak !”

Şimdi kendinizi bir hazinenin içinde hissedebilirsiniz. Oyun geliştirmek istediğinizde oyun modülleri, internet sitesi yapmak istediğinizde bunla ilgili modüller yardımınıza koşacak.



Hadi modüller üzerinde biraz pratik yapalım. Python'da sıklıkla kullanılan modüllerden biri random yani rastgele modülüdür. Çünkü şöyle düşünün. Bir oyun yaptınız. Oyunda bir karakterin rastgele ekranın bir yerinde çıkmasını istiyorsunuz. O zaman ne yapacaksınız? Tabi ki rastgele modülünü kullanacaksınız. Biz de küçük bir oyun yapalım. Makine aklından bir sayı tutup bize söylesin.

```
saved [Share] [Run] [Refresh] [Close]
Add new file
1 import random
2 print ("Aklımdan tuttuğum sayı:",
3       random.randint(0, 100))
4
5
```


```
Python 3.6.1 (default, Dec 20
[GCC 4.8.2] on linux
Aklımdan tuttuğum sayı: 82
```

Ne yaptığımıza tekrar bakalım. import random ile rastgele modülünü programımıza ekledik. Evet şimdi random içerisindeki bir sürü fonksiyon programımızda çalışabilir. örneğin **print(random.random())** şeklinde bir kod çalıştırarak random modülünün random fonksiyonunu çağırırsak ekrana 0 ile 1 arasında rastgele bir şey yazar. Örneğin: 0.12334223 gibi.

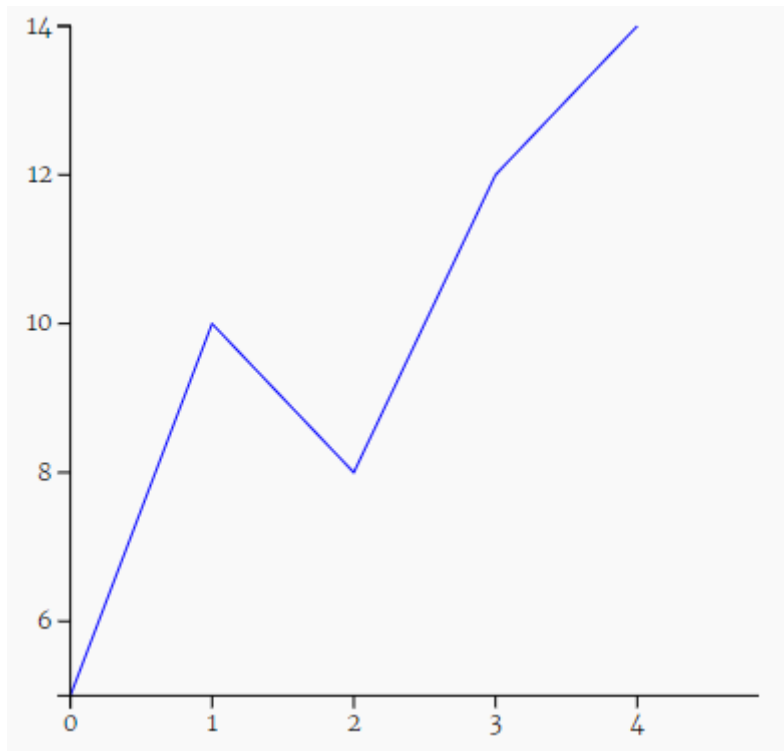
Fakat aklında bir sayı tut denilince herhalde arkadaşınız aklımdan tuttuğum sayı 0.1334 demez. Eğer öyle diyorsa oradan yavaşça uzaklaşın :) Arkadaşınız muhtemelen 0,1,2,4,6,42 gibi bir sayı söyleyecektir. Bizim programımız da öyle yapacak. Bunun için randint fonksiyonunu kullanıyoruz randint "Random Integer" ifadesinin kısaltması. Yani "Rastgele Tam Sayı" parantez içerisinde kullandığımız 0,100 ise 0 ile 100 arasında bir sayı tutmasını

sağlıyor. Programı ilk çalıştırdığımızda aklımdan tuttuğum sayı 82 dedi. Bir sonraki çalışmada 82 ile tamamen alakasız 25 de diyebilir. Tekrar 82 de diyebilir. Adı üstünde "rastgele". Random modülü Python içerisinde kurulu olarak gelen bir modül.

Şimdi başka bir örnekle devam edelim. Örneğin şöyle bir sorumuz olsun. Okulun futbol takımı her yıl 15 maça katılıyor. Bu maçların kaç tanesini yendiğini ekrana grafik olarak yazdıralım. Bizim çizimler için kullandığımız matplotlib.pyplot isminde bir modülümüz var. Bu modülü kullanacağız. Fakat bu modülün ismi biraz uzun. Her defasında yazması zor. Bu yüzden; **import matplotlib.pyplot as cizim** . Yani matplotlib.pyplot modülünü içeri aktar ama cizim ismiyle kullanayım. Son kısımdaki **as cizim** bu işe yarıyor. Eğer cizici ismiyle kullanacak olsaydım **import matplotlib.pyplot as cizici** diyecektim. Evet, devam edelim. Her modülün kendi fonksiyonları olduğunu söylemiştik. Burada kullanacağımız modülün **plot**, **show** gibi fonksiyonları var. Plot bir grafik çizerken, show da bu grafiği ekranda gösteriyor.

```
< > main.py +   
1  
2  
3 import matplotlib.pyplot as cizim  
4  
5 cizim.plot([0,1,2,3,4,5], [5,10,8,12,14])  
6 cizim.show()  
7  
8
```

Kodumuzda da bunları kullandık. cizim.plot diyerek önce X eksenini(soldan sağa doğru olan), sonra Y eksenini (aşağıdan yukarıya doğru olan) sayıları girdik. cizim.show() ile bunları ekranda gösterdik.



İlk yıl 5 maç kazanırken son yıla doğru tüm maçları kazanmaya başlamışız neredeyse. Helal olsun bizim okula. :)

Buradaki grafiğin aralıkları, rengi gibi bir çok şey değiştirebileceğinizi unutmayın. Sadece örnek olması için modülümüzü kullanarak bir grafik çizdik.

-“Buğra abi kızma da bir şey diyeceğim. Ben böyle grafikler falan değil de oyun yapmak istiyorum. Nasıl yapacağız ?“

-”Tamam şimdilik oyun yapalım ama büyüüp iyi bir yazılımcı olduğunda grafikleri de çizmen gerekeceğini unutma. Anlaştık mı?“

Benim çocukken en sevdiğim çizgi film Ninja Kaplumbağalar’dı. Her gün evdeki video kaseti bir gün izlerdik. O zamanlar Youtube yoktu. Evinizde 3,4 tane video kaset olurdu. Çevirip çevirip onları izlerdiniz. Siz rahatsınız, hayat size güzel. :)

Nereden geldik buraya diyorsanız hemen söyleyeyim. Şimdi öğreneceğimiz modülün adı Turtle. Turtle İngilizce’de kaplumbağa anlamına geliyor. Turtle özellikle “Oyun yapmak istiyorum!” diyen öğrenciler için kullanılan bir modül.

Hadi incelemeye başlayalım. Kaplumbağamın adı bilgin olsun.

```
< > main.py
1 import turtle
2
3 bilgin = turtle.Turtle()
4
5 bilgin.forward(50)
```



Ekran görüntüsü :

İlk önce import turtule ile kaplumbağa kütüphanesini ekledim.

bilgin = turtle.Turtle() diyerek bilgin isminde bir kaplumbağa oluşturdum. bilgin.forward(50) diyerek kaplumbağamı 50 pixel ileriye götürdüm. Kodumu çalıştırdığımda bir ok sol taraftan başlayarak hızlıca sağa doğru gitti.

-“Buğra abi kaplumbağa nerede?“

-”Önce kaplumbağanın nasıl hareket ettiğini anlayacağız. Sonra ok yerine kaplumbağa resmi koyacağız ve kaplumbağa hareket edecek.“

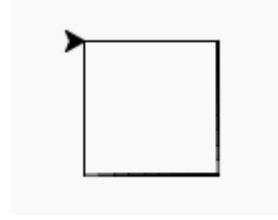
Kodumuzu biraz geliştirdik. Şimdi kaplumbağamız 50 adım ileri gidiyor, sonra sağa dönüp 90 adım gidiyor. Sonra tekrar ileri 50 adım gidiyor ve tekrar sağa dönüp 90 adım gidiyor. Bunu 4 kez yaptığında bir kutunun etrafında döndüğünü görüyoruz.

```
import turtle

bilgin = turtle.Turtle()

for i in range(4):
    bilgin.forward(50)
    bilgin.right(90)
```

Ekran Görüntüsü:



Şimdi Turtle modülünü kullanarak ilk oyunumuzu geliştirelim.

```
> main.py + [icon]
1
2 import turtle
3
4 ekran = turtle.Screen()
5
6 ekran.setup(400, 400) #boyutları belirledik
7
8 ekran.bgpic("space.jpg") #arkaplan resmi seçtik
9
10
```


Burada ne yaptık ? Önce turtle modülümüzü ekledik. Ardından ekran isminde bir değişken oluşturduk. Bu değişken ekran = turtle.Screen() ifadesiyle oyunumuzun sahnesi oldu. Screen İngilizce’de ekran, sahne anlamına geliyor. Peki sahnenin boyutları ne olacak ? Çünkü oyun açıldığında ne kadarlık bir alanda olacağını biz seçiyoruz.

ekran.setup(400,400) diyerek bu sahnenin boyutlarını belirledik. 400’e 400 pixellik bir alanda oldu. Ardından ekran.bgpic(“space.jpg”) diyerek oyunumuzun arka planını seçtik. Aslında bilgisayarımızda bulunan bir resim dosyasını çalıştırdık. Bu resim dosyasının kodumuz ile aynı klasörde olması çok önemli. Buna dikkat ediyoruz. Buradaki bgpic ise “Background Picture” yani İngilizce’deki “Arkaplan Resmi” ifadesinin kısaltılmış hali.

Ekran görüntümüze bakalım.

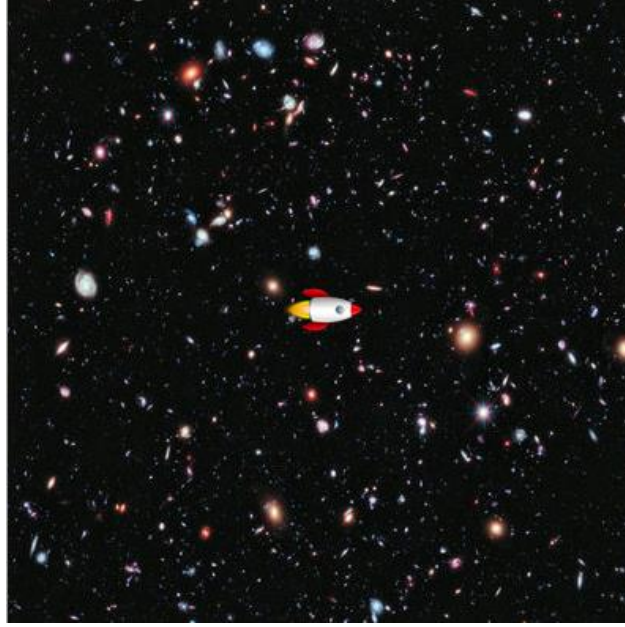


Evet. Biz uzay var ama içerisinde hareket eden bir şey yok. Hemen uzay oyunumuza bir şeyler ekleyelim.

```
main.py +   
  
import turtle  
ekran = turtle.Screen()  
ekran.setup(400, 400) #boyutları belirledik  
ekran.bgpic("space.jpg") #arkaplan resmi seçtik  
ekran.addshape("rocketship.png")  
turtle.shape("rocketship.png")
```

İki satır kod ekledik. `ekran.addshape("rocketship.png")` ve `turtle.shape("rocketship.png")` yani ekrana rocketship isiminde bir resim ekle. Ardından turtle yani ekrandaki hareketli nesnenin resmine bunu ayarla. Shape, İngilizce'de şekil anlamına geliyor.

Ekran görüntümüz aşağıdaki şekilde oldu.



Ekranında bir roketimiz var ama hareket etmiyor. Hareket fonksiyonu yok. Aaa fonksiyon mu ? Hemen fonksiyonları yazalım öyleyse. Roketimiz ileriye, geriye gidecek. Sağa ve sola dönecek. Ayrıca her tuşa basıldığında ne kadar hareket edeceğini yazacağız. Bunlar için aşağıdaki kod blogunu ekliyoruz.

```
2
3 hareket_et = 10
4
5
6 def ileri():
7     turtle.forward(hareket_et)
8
9 def geri():
10    turtle.backward(hareket_et)
11
12 def sol():
13    turtle.left(hareket_et)
14
15 def sag():
16    turtle.right(hareket_et)
17
```


hareket_et kısmındaki 10 bizim 10 pixel hareket edeceğimizi söylüyor. Diğer 4 fonksiyonumuz ise ileri,geri,sol ve sağa gidişleri tanımlıyor. Burada turtle.forward, turtle.backward , turtle.left ve turtle.right fonksiyonları turtle modülümüzün içerisinde gelen fonksiyonlar. Eğer bir oyun geliştirecek olursak turtle modülünü açıp fonksiyonların isimlerine de bakmamız gerekiyor.

Evet artık roketimiz hareket edebilir. Fakat hareket etmesi için bir tuş tanımlamadık. Oyun başladığında hangi tuşa basınca yukarı gidecek. “Yukarıuuuuuu diye bağırınca gitmeyecek değil mi?” hadi tuşlarımızı ayarlayalım.


```
ekran.onkey(ileri, "Up")
ekran.onkey(geri, "Down")
ekran.onkey(sol, "Left")
ekran.onkey(sag, "Right")
ekran.listen()
```

İşte bu kadar basit. Ekran bizim sahnemizdi. Sonrasındaki onkey ifadesi bir tuşa basıldığında olacakları belirleyen fonksiyonumuz. Şu şekilde çalışıyor.

ekran.onkey("HANGİ FONKSİYONU ÇALIŞTIRALIM", "HANGİ TUŞA BASILINCA ÇALIŞSIN") . Biz de buna uygun olarak ileri fonksiyonu için "Up" yani yukarı tuşunu girdik. Gönül oraya Up değil Yukarı yazmak isterdi fakat ne yazık ki klavye tuşlarının İngilizce karşılıkları geçerli. İnşallah sizin yazacağınız dillerde buralarda da Türkçe olacak. Aynı şekilde 4 tuşu girip en sonda ekran.listen() diyoruz. Yani ekranı dinle. Bu şekilde yeni bir tuşa basıldığında da roketimiz hareket edebiliyor.

```
< > main.py + 
1
2 import turtle
3
4 ekran = turtle.Screen()
5
6 ekran.setup(400, 400) #boyutları belirledik
7
8 ekran.bgpic("space.jpg") #arkaplan resmi seçtik
9
10 ekran.addshape("rocketship.png")
11 turtle.shape("rocketship.png")
12
13 hareket_et = 10
14
15
16 def ileri():
17     turtle.forward(hareket_et)
18
19 def geri():
20     turtle.backward(hareket_et)
21
22 def sol():
23     turtle.left(hareket_et)
24
25 def sag():
26     turtle.right(hareket_et)
27
28 ekran.onkey(ileri, "Up")
29 ekran.onkey(geri, "Down")
30 ekran.onkey(sol, "Left")
31 ekran.onkey(sag, "Right")
32 ekran.listen()
```

Result



Vee roketimiz uçuşa hazır. Turtle modülünü kullanarak ilk oyununuzu yazdınız. Kendinizi alkışlamayı unutmayın.

Dikkat ettiniz mi ? Her modülün kendine özel fonksiyonları var. "Buğra abi ben bütün modüllerdeki bütün fonksiyonları nasıl öğreneyim!" diye düşünmeyin. Programda hangi modüle ihtiyaç duyarsanız onun kullanma kılavuzu gibi olan dokümanını açıp fonksiyonlara bakarak kodunuzu yazabilirsiniz. Ayrıca bir modülü çok kullandığınızda içindeki fonksiyonlara da aşina olacaksınız eminim.

Bölüm 14: Dosya İşlemleri

Python'da bir çok işlem yaptık. Kimi zaman sayılarla , kimi zaman kelimelerle oynadık. Fonksiyonlar tanımladık, modülleri kullandık. Fakat önemli bir sorunumuz var. Bu işleri yaptıktan sonra elde ettiğimiz veriyi nerede saklayacağız ? Programın çıktı ekranında olan şeyler, program kapanınca gidecek.

İşte burada yardımımıza dosyalar koşuyor. Programımızı kullanarak istediğimiz bilgileri bir dosyaya kaydedebiliyor, dosya içeriğini değiştirebiliyor, dosyadan okuma yapabiliyor veya dosyayı silebiliyoruz.

Adım adım gidelim. İlk olarak bir dosya oluşturmamız lazım. Bunun için;

DEGISKENISMIM = open("DOSYAISMI.TXT, DOSYAKIPI) şeklinde bir yazım dili kullanıyorum. Örneğin dosyam isimli bir değişkenim olacak. Bu "dosyam.txt" isimli bir dosya oluşturacağım. Bu dosyaya yazma iznim olacak diyelim.

Bunun için; dosyam = open("dosyam.txt","w") kodunu kullanmam yeterli oluyor. Mutlaka aklınıza takılmıştır. Dosya ismi tamam da dosya kipi ne demek ?

Parametre	Anlamı
r	Salt Okuma kipi
w	Yazma kipi
a	Sonuna ekleme kipi
b	İkili (Binary) kip
t	Salt metin kipi (ön tanımlı)
+	Dosya güncelleme kipi (hem okuma hem yazma)

[1] <https://www.python.tc/python-dosya-islemleri/>

Dosya kipi biz dosyayı açarken ne yapmak istediğimizi tanımlayan bir durum. Eğer w yazarsan dosyayı yazmak üzere açıyoruz, r dersek varolan bir dosyayı okumak istiyoruz. Tabloda gördüğünüz gibi benzeri bir kaç kipimiz daha var. Peki dosyayı oluşturmak için **dosyam = open("dosyam.txt","w")** kodunu kullandık. Şimdi dosyanın içerisine bir şeyler yazalım. İngilizce'de yaz "write" demek. Python'da da aynı **dosyam.write("İstediğimiz şeyleri buraya yazıyoruz.")** diyerek dosyanın içerisine istediğimiz ifadeyi yazabiliyoruz. Dosyayı açtık, istediklerimizi yazdık. Dosyayı nasıl kapatacağız. Bu da İngilizce'de kapat anlamına gelen "close" kelimesiyle oluyor. **dosyam.close()** dediğimizde dosyamız kapanıyor.

Şimdi programımızı yazalım.

```
dosyaislemleri > main.py >
main.py x dosyaoku.py x ekleme.py x giris.txt x bolumoku
1 dosyam = open("dosyam.txt","w")
2
3 dosyam.write("Merhabalar !\n")
4 dosyam.write("Bu yeni bir satir!\n")
5 dosyam.write("Iste yeni bir satir!\n")
6 dosyam.write("Hadi bu isi bitirelim.\n")
7
```

Program dosyayı oluşturdu. Bir şeyler yazdı ve kapattı. Burada satır sonlarındaki "\n" kısmı alt satıra geçmeye yarıyor. Dosyanın okunaklı olmasını sağlıyor. Kullanmasak da programımız çalışır.

Şimdi bu oluşturulan dosyam.txt dosyasının içeriğini okuyalım. Benzer şekilde bilgisayardaki başka bir metin dosyasının içeriğini de okuyabileceğinizi unutmayın.

Okumak İngilizce'de "read" demek değil mi ? Öyleyse biz de "read" diyeceğiz. Okuma kipimizi de "r" olarak tabloda yazmıştık. Programımız dosyayı okuyacak. Ardından okuduğunu print ile çıktı ekranına yazdıracak. İşte bu kadar basit.

```
1 dosya = open("dosyam.txt", "r")
2 print (dosya.read())
3
```

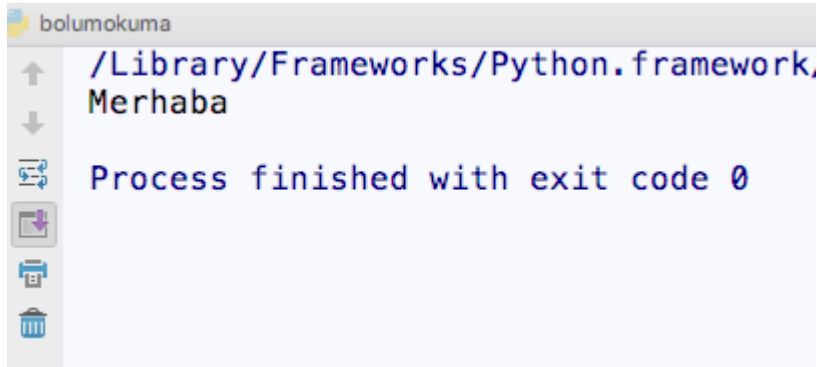
```
in: kullanicidanalma dosyaoku
/Library/Frameworks/Python.framework/Ve
Merhabalar !
Bu yeni bir satir!
Iste yeni bir satir!
Hadi bu isi bitirelim.
```

Çıktımız görüldüğü gibi satır, satır oldu.

Dosyayı oluşturduk, okuduk. Başka neler yapabiliriz ? Mesela dosyanın içinden sadece istediğimiz kadar kısmı okuyalım. Örneğin ilk 7 karakteri okuyalım. Bizim oluşturduğumuz dosyada ilk 7 karakter "merhaba" şeklinde. Bunun için print ile ekrana yazdırma yaparken dosya.read ifadesinin içine 7 şeklinde bir değer atıyoruz. Bu sayede sadece ilk 7 karakter okunuyor.

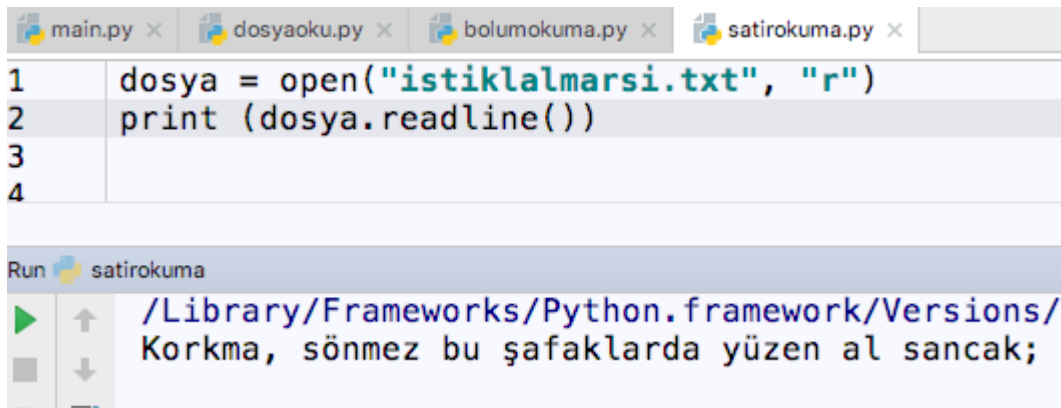
```
main.py x dosyaoku.py x bolumokuma.py x
1 dosya = open("dosyam.txt", "r")
2
3 print (dosya.read(7))
4
5
```


Ekran çıktımız aşağıdaki şekilde oluşuyor.



```
bolumokuma
↑ /Library/Frameworks/Python.framework,
↓ Merhaba
Process finished with exit code 0
```

Peki diyelim ki elimizde bir kaç satırdan oluşan bir metin var. Biz sadece ilk satırı okumak istiyoruz. Satırın kaç karakter olduğunu da bilmiyoruz. Ne yapacağız ? Python bunun için de bir kolaylık yapmış. `readline()` isimli metodu kullanarak ilk satırı ekrana yazdırabiliyorsunuz. Biz örnek olarak İstiklal Marşımızı bir dosyaya kaydettik. Adına da "istiklalmarsi.txt" dedik. Ardından "r" dosya kipiyle marşımızı okuduk. `readline()` metoduyla ilk satırı olan "Korkma, sönmez bu şafaklarda yüzen al sancak;" kısmını ekrana yazdırdık.



```
main.py x dosyaoku.py x bolumokuma.py x satirokuma.py x
1 dosya = open("istiklalmarsi.txt", "r")
2 print (dosya.readline())
3
4

Run satirokuma
↑ /Library/Frameworks/Python.framework/Versions/3:
↓ Korkma, sönmez bu şafaklarda yüzen al sancak;
```

"Güzel ama ben dosyaya sadece koda önceden belirttiğim şeyleri girmek istemiyorum! Mesela program bana bir soru sorsun, ben cevabını vereyim. Dosyaya da o yazılsın." diyor olabilirsiniz. Çok da haklısınız. Bunu yapalım. Kullanıcıdan bilgi almak için `input` fonksiyonunu kullanıyorduk.

İlk olarak `giris` isimli değişkenimize `input` ile bilgi alacağız. Ardından `giris.txt` isminde bir dosyayı "w" yani yazma kipinde oluşturup içerisine `giris` değişkenini yazacağız. Ardından da dosyayı kapatacağız.

```
1 giris = input("Kaydedeceğiniz bilgiyi yazın: ");
2
3 dosyam = open("giris.txt","w")
4
5 dosyam.write(giris)
6 dosyam.close()
7
8
```

```
Run kullanicidanalma
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5
Kaydedeceğiniz bilgiyi yazın: Merhaba bu bilgiyi kendim girdim.
Process finished with exit code 0
```

Yukardaki programımızda bize "Kaydedeceğiniz bilgiyi yazın:" denildi. Biz de cevap olarak: "Merhaba bu bilgiyi kendim girdim." dedik. Enter tuşuna bastığımızda bu yazdığımız bilgi "giris.txt" dosyasının içine yazıldı.

Peki önceden mevcut bir dosyaya sadece ekleme yapmak istediğimizde ne yapıyoruz ? Bu sorunun cevabında da yine bir dosya kipi olan "a" yardımımıza yetişiyor. İngilizce'de ekle anlamına gelen "add" kelimesinin baş harfi olan "a" dosyamıza ekleme yapmamızı sağlıyor.

```
main.py x dosyaoku.py x ekleme.py x giris.txt x bolumokuma.p
1 dosya = open("giris.txt", "a")
2
3 dosya.write("Bu bilgiyi kendim ekledim.")
4
```

Programımızda olduğu gibi "a" kipiyle ekleme modunu açıp, write metoduyla istediğimiz ifadeyi ekleyebiliyoruz. Eğer istersek bu girişi input fonksiyonunu kullanarak kullanıcıdan da alabiliriz. Ekran çıktımız aşağıdaki gibi oluyor.

```
main.py x dosyaoku.py x ekleme.py x giris.txt x bolumokuma.py x satirokuma.py x
1 Merhaba bu bilgiyi kendim girdim.Bu bilgiyi kendim ekledim.
```

-"Bugra abi bir şey diyeceğim."
-"Efendim, noldu ?"
-"Giriş değil de çıkış yapsaydık keşke o ismi."
-"Niye, hayırdır ?"
-"Bilmem canım öyle istedi. Yapamıyor musun yoksa :D"
-"Tamam yapalım."

Dosya isimlerini değiştirmek için os isimli bir modülü kullanmamız gerekiyor. OS "Operation System" yani İşletim Sistemi kelimesinin İngilizce karşılığının baş harflerinden oluşuyor. Adından da anlaşılacağı üzere işletim sistemi ile ilgili işlemleri yapıyoruz. Dosya da sonuçta bu sistemin içerisinde bir yere kaydediği için os modülü ile ilgili işlemleri yapıyoruz.

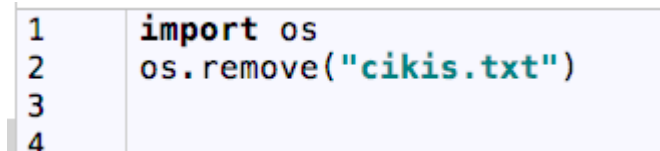


```
1 import os
2 os.rename( "giris.txt", "cikis.txt" )
3
4
5
6 |
```

import os ifadesiyle modül ekledikten sonra yapmamız gereken son derece basit. os.rename ("ÖNCEKİSİM", "SONRAKİSİM") yapıyoruz. Program çalışınca tak diye dosya adımız değişiyor.

- "Buğra abi şimdi silelim mi dosyayı?"
- "Niye?"
- "Kalabalık yapmayın, bilgisayar yavaşlıyor."
- "Küçük bir metin dosyası ne kadar yavaşlatacak bilgisayarı?"
- "Yoksa silemiyor musun?"
- "Tamam yapalım."

Dosyaları silmek için yine os modülünü kullanıyoruz. Ardından da İngilizce'de silmek anlamına gelen "remove" metodunu çağırarak dosyamızı siliyoruz. Kullanım yöntemimiz: os.remove("SİLİNECEKDOSYAİSMİ) şeklinde gayet basit.



```
1 import os
2 os.remove("cikis.txt")
3
4
```

Programımız çalıştığında dosyamız silinmiş oldu.

Evet her şey çok güzel gitti fakat bizim dosya okuma işlemlerinde karşılaştığımız bazı küçük sorunlar var. Diyelim ki siz bir arkadaşınızın eline bir kitap verdiniz ve "Hadi bunu oku." dediniz. Arkadaşınız da sayfaları tek tek çevirerek okudu ve en sona geldi. En son kelimeyi de söyleyip bitti dedi. Fakat yazıların olduğu en son sayfada kaldı. "Kitabı tekrar oku." Dediğinizde okuması için tekrar en başa dönmesi gerekir değil mi ? İşte Python'da dosya işlemlerinde de aynı şeyi yapmamız lazım. Sorunuza bir kod ile daha yakından bakalım.

```
main.py x dosyaoku.py x ekleme.py x giris.txt x bolumok
1 dosya = open("dosyam.txt", "r")
2 print (dosya.read())
3 print ("Tekrar okuyalım.")
4 print (dosya.read())
5

Run: kullanicidanalma dosyaoku
/Library/Frameworks/Python.framework/Versio
Merhabalar !Bu yeni bir satir!Iste yeni bi
Tekrar okuyalım.

Process finished with exit code 0
```

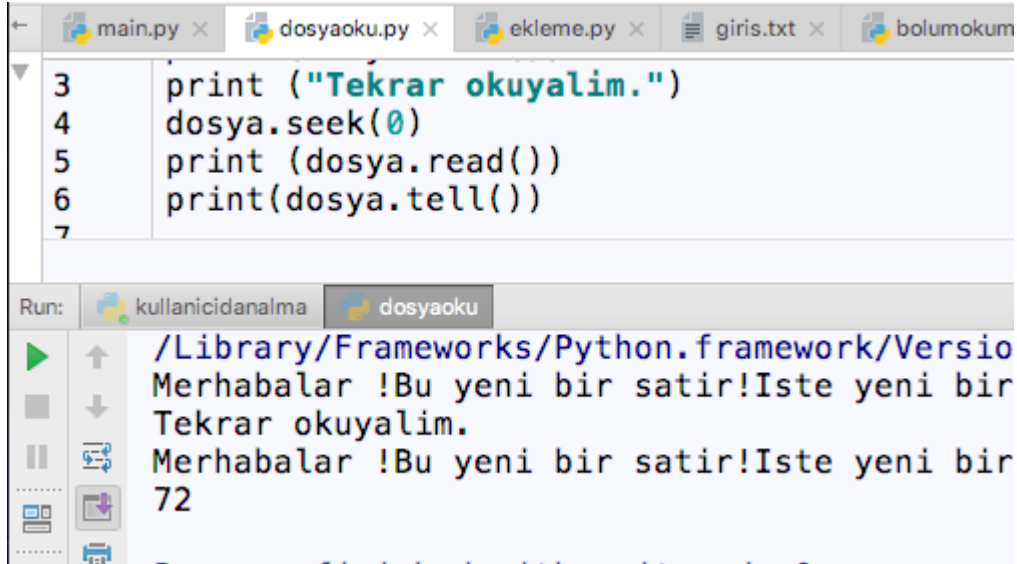
Kodumuzda "dosyam.txt" isimli dosyayı okuduk ve ekrana yazdırdık. Bu bizim önceden belirlediğimiz "Merhabalar! Bu yeni bir satir!" şeklinde başlayan metnimizdi. Sorunsuz yazıldı. Ardından "Tekrar okuyalım." diyerek tekrar okumak istedik. Fakat çıktı ekranında görüldüğü gibi "Tekrar okuyalım" ifadesinin altı boş kaldı. Tıpkı kitabın son sayfasında kalan çocuk gibi programımız yeni bir şey okuyamadı. Bu sorunu çözmek için seek isimli metodu kullanıyoruz. Seek metodu içerisine bir sayı alıyor. Örneğin seek(0) dersek en başa gidiyor. Elbette burada sayfa değil byte yani bir veri büyüklüğü söz konusu. Eğer seek(10) dersek dosyanın 10. Baytlık kısmına gidiyor. Biz byte büyüklüğü ile pek uğraşmayacağız ama seek(0) ile dosyanın başına gitmemiz işimizi kolaylaştıracaktır. Kodumuzu çalıştıralım.

```
main.py x dosyaoku.py x ekleme.py x giris.txt x bolumokuma.py
1 dosya = open("dosyam.txt", "r")
2 print (dosya.read())
3 print ("Tekrar okuyalım.")
4 dosya.seek(0)
5 print (dosya.read())

Run: kullanicidanalma dosyaoku
/Library/Frameworks/Python.framework/Versions/
Merhabalar !Bu yeni bir satir!Iste yeni bir sa
Tekrar okuyalım.
Merhabalar !Bu yeni bir satir!Iste yeni bir sa

Process finished with exit code 0
```

Görüldüğü gibi bu sefer "Merhabalar!" ile başlayan yazımız iki kez ekranda yer aldı. Seek metoduna benzer bir metod ise Tell. Tell İngilizce'de "Söylemek" anlamına geliyor. Kitap okuyan arkadaşınıza "Kaçınca sayfadasin söyle bakayım?" dersiniz ya. İşte tell metodu ile Python'da "İmleç yani işaretçi dosyanın kaçınca byte'ında söyle bakayım." diyorsunuz. Hadi test edelim.



```
3 print ("Tekrar okuyalim.")
4 dosya.seek(0)
5 print (dosya.read())
6 print(dosya.tell())
7
```

Run: kullanicidanalma dosyaoku

```
/Library/Frameworks/Python.framework/Versio
Merhabalar !Bu yeni bir satir!Iste yeni bir
Tekrar okuyalim.
Merhabalar !Bu yeni bir satir!Iste yeni bir
72
```

Kodumuzun en altında 72 yazdı. Yani 72 byte değerinde bir dosya oluşturmuşuz. Çünkü read ile dosyanın tamamını okuduk. 72 bizim için sona gelindiğinde ulaşılan byte boyutunu söylüyor.

Dosyalar kısmının da sonuna geldik. Artık programlarınızda yaptığınız işlemleri dosyalara yazdırabilir, önceden oluşturulmuş dosyalardaki verileri dilediğiniz gibi kullanabilirsiniz 😊

Bölüm 15: Python Orkestrası

Uzun yolculuğumuzun sonucunda buraya kadar geldik. En başta şöyle bir şey konuşmuştuğ. Bir orkestrayı oluşturur gibi çalışacağız. Sırasıyla tüm müzik aletlerini tanıyacağız. Ardından bu aletleri kullanarak müzik yapmaya başlayacağız. Biz de bunu yaptık. Döngüler orkestramızın flütüydü, değişkenler kemanı, modüller ise davulu. Bir şey itiraf etmeliyim. Müzik aletlerini tek tek öğrenmek orkestradaki müziği dinlemek kadar zevkli değil. İşte biz de yolun bu zor kısmını aştık. Şunu unutmayın. Bir müzik aletini öğrenirken onunla ilgili tüm detayları öğrenemeyebilirsiniz, ama aleti çaldıkça artık onu çok iyi öğrenirsiniz. Burada da örneğin listeler konusundan bahsederken belki tüm küçük detayları öğrenmediniz ama kod yazdıkça bu detayları da göreceksiniz. Benzer şekilde biri davul çalmayı öğrenen öğrenciler bunu farklı müzikleri çalarak öğrenebilir. Burada da modülleri öğrenirken "turtle" modülünden örnek kullandık. Başka bir kaynaktan ise farklı bir modülden örneklendirme yapılabilir. Bunlar sizin programları sorunsuz yazmanıza engel değil.

Peki şimdi ne yapacağız ? Şimdi öğrendiklerimizi pekiştireceğiz. Farklı müzik aletleriyle yapılan müzik gibi biz de konularda öğrendiğimiz farklı fonksiyonları,metotları,değişkenleri birleştirerek programlar yazacağız. Hadi başlayalım.

Örnek Uygulamalar

Üçgende Kenar Hesaplama

Bir ABC üçgeninde a ve b kenarlarının uzunlukları toplamı c kenarının uzunluğundan küçük olamaz. Aynı şekilde a ve b kenarlarının uzunlukları farkı c kenarından büyük olamaz. Örneğin a=5 cm ve b=10 cm ise c kenarının uzunluğu $10-5 < c < 10+5$ yani $5 < c < 15$ aralığında olmalıdır. Programınızda kullanıcıdan aldığınız iki kenara göre üçüncü kenarın uzunluğunun hangi aralıkta olacağını hesaplayarak ekranda gösterin.

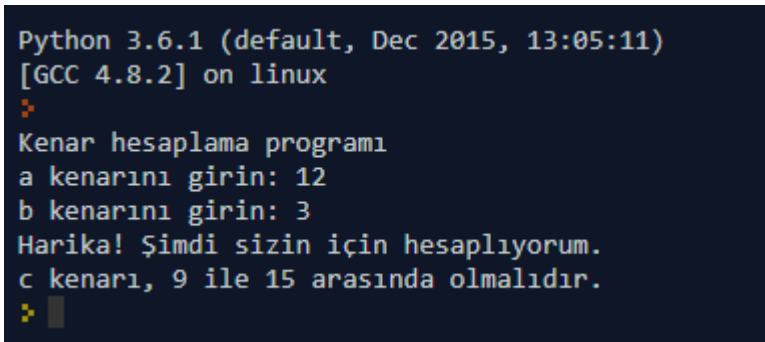
```
print ("Kenar hesaplama programı")
a = int(input("a kenarını girin:"))
b = int(input("b kenarını girin:"))
print("Harika! Şimdi sizin için hesaplıyorum.")

if a>b:
    altsinir = (a-b)
else:
    altsinir = (b-a)

ustsinir = (a+b)

print("c kenarı,", altsinir , "ile", ustsinir, "arasındadır.")
```

Programımızın ekran çıktısı şu şekilde.



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
>
Kenar hesaplama programı
a kenarını girin: 12
b kenarını girin: 3
Harika! Şimdi sizin için hesaplıyorum.
c kenarı, 9 ile 15 arasında olmalıdır.
>
```

Burada iki noktaya dikkat etmeliyiz. İlki kullanıcıdan input ile değişken alırken bu değişkenin türünün String olduğu. Biz bu değişkeni int() parantezine alarak sayıya çevirdik. İkincisi de alt sınır için iki ihtimal vardı. Kullanıcının girdiği a ve b kenarlarından hangisinin büyük olduğunu bilmiyorduk. Bu yüzden bir if döngüsü kullanarak kontrol yaptık.

Akıllı Termometre

Sıcaklığı ölçmek için farklı termometreler kullanılır. Bizim "Hava kaç derece?" dediğimizde verdiğimiz 20 derece, 30 derece gibi cevaplar aslında Celcius termometresine göredir. Fakat benzer şekilde Fahrenheit, Kelvin ve Reomür termometreleri de vardır. Bunların her biri farklı şekillerde sıcaklığı hesaplar. Eğer siz hava sıcaklığını Celcius cinsinden, yani gündelik hayatta söylediğimiz gibi, biliyorsanız basit matematiksel işlemlerle diğer türlerden de

bulabilirsiniz. Programınızda kullanıcıdan hava sıcaklığını alın. Ardından bu sıcaklığı ekrana 4 farklı termometre için de yazın.

```
print ("Akıllı Termometre Programı")
celsius = int(input("Oda sıcaklığını girin:"))

fahrenheit = (celsius * 1.8) + 32
kelvin = celsius + 273
reomur = celsius * 0.8

print("Tüm Termometre türleri için hesaplama yapıldı.\n")
print("Celsius:", celsius , "\n")
print("Fahrenheit:", fahrenheit , "\n")
print("Kelvin:", kelvin , "\n")
print("Reomur:", reomur , "\n")
```

Ekran çıktımız şu şekilde:

```
Akıllı Termometre Programı
Oda sıcaklığını girin: 27
Tüm Termometre türleri için hesaplama yapıldı.

Celsius: 27

Fahrenheit: 80.6

Kelvin: 300

Reomur: 21.6
```

Burada kullanılan matematiksel hesaplamalara baktığımızda Kelvin değeri için Celsius'a 273 eklediğimizi, Reomur için 0.8 ile çarptığımızı görebilirsiniz. Fahrenheit'ın işlemi bunlara göre kısmen daha karışık fakat onda da değerimizi 1.8 ile çarpıp üzerine 32 eklediğimizde değere ulaşıyoruz. Bu işlemlerin bizle alakası yok, bilimsel formülleri hazır olarak kullandık. Siz de ihtiyacınız olan formülleri bu şekilde kullanabilirsiniz. Örneğin bir başka termometre türü olan Rankine için siz de bir dönüşüm yapabilirsiniz.

Metin Temizleyici

Yapay Zeka çalışmaları sırasında makinelere çeşitli metin kalıpları öğretilirken karşılaşılan zorluklardan biri o metin üzerinde istenmeyen özel karakterler olabilmesidir. Örneğin: "Merhaba" cevabına yapay zeka "Sana da merhaba" şeklinde cevap verebilirken "Merhaba%\$" gibi bir yazıda kafası karışabilir ve cevap veremeyebilir. İşte bu yüzden bir program geliştireceğiz. Programımız bize yazılan metnin içerisindeki özel karakterleri temizleyerek, yazının temiz halini ekranda gösterecek.

```

temizlenecek karakterler = '!()-[]{};: ",<>./?@$%^&* _~+'

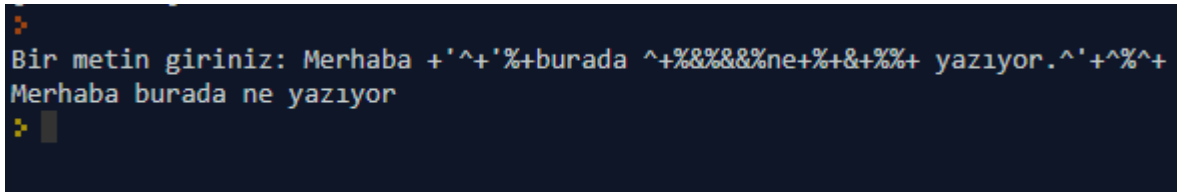
metin = input("Bir metin giriniz:")

temizlenmismetin = ""
for karakter in metin:
    if karakter not in temizlenecek karakterler:
        temizlenmismetin = temizlenmismetin + karakter

print(temizlenmismetin)

```

Kodumuzun ekran çıktısı ise şu şekilde:



```

> Bir metin giriniz: Merhaba +'^'+burada ^+%&&&&ne+%&+%+% yazıyor.^'+^'+
Merhaba burada ne yazıyor
>

```

Görüldüğü gibi karakterlerimizi rahatça temizledik. Kodumuzun nasıl çalıştığına tekrar dönelim. İlk olarak temizlemek istediğimiz özel karakterleri "temizlenecek karakterler" isimli bir değişkenin içerisine koyduk. Ardından kullanıcıdan bir metin girmesini istedik. Kullanıcıdan metni aldıktan sonra şuna baktık. Kullanıcının girdiği metinde bu karakterler var mı? Tıpkı bir süzgeçten geçirir gibi bu karakterleri dışarıda tuttuk. Kalan karakterleri sırasıyla, başlangıçta boş olarak oluşturduğumuz, temizlenmismetin isimli değişkenimize ekledik. Son olarak da bu değişkeni ekrana yazdırdık.

Kelime Sıralayıcı

Kullanıcıdan alınan metinde geçen kelimeleri tıpkı sözlükte olduğu gibi a dan başlayarak z ye gidecek şekilde düzgün bir sırada gösteren bir program geliştirin.

```

print ("Sözcük sıralama programı:")
cumle = input("Bir metin girin: ")

kelimelistesi= cumle.split()

kelimelistesi.sort()

print("Listemiz sıralanmış şekilde oluşturuldu:")
for kelime in kelimelistesi:
    print(kelime)

```

Ekran çıktımız şu şekilde oluştu.


```
Sözcük sıralama programı:  
Bir metin girin: merhaba burada bir metin var. Bu metinde kelimeler karışık halde.  
Listemiz sıralanmış şekilde oluşturuldu:  
Bu  
bir  
burada  
halde.  
karışık  
kelimeler  
merhaba  
metin  
metinde  
var.
```

Programımıza tekrar bakalım. Öncelikle cumle isminde bir değişkene kullanıcının girdiği metni atadık. Ardından split() metoduyla girilen metni boşlukları esas alarak kelimelere ayırdık ve bir listeye atadık. Sonrasında bir başka metodumuz olan sort() ile kelimeleri sıraladık. Kelimelistesi adını verdiğimiz bu listedeki elemanları sırasıyla ekrana yazdırdık.

Duygu Analiz Programı

Sosyal ağların gelişmesiyle beraber en çok ihtiyaç duyulan şeylerden biri duygu analiz programları oldu. Bu programlar bir Tweet'in hangi duygularla atıldığını, olumlu mu olumsuz mu olduğunu anlamaya çalışıyor. Biz de Python'da minik bir duygu analiz programı yapacağız. Önceden olumlu ve olumsuz kelimeleri öğrettiğimiz program, kullanıcının girdiği cümlelerin olumlu mu yoksa olumsuz mu olduğunu anlamaya çalışacak.

```

def duyguanalizi():
    olumlu = ["harika","muhteşem","süper"]
    olumsuz = ["berbat","iğrenç","kötü"]

    cumle = input("Bir cümle girin:")

    kelimelistesi = cumle.split()

    olumlukesisim= list(set(olumlu) & set(kelimelistesi))
    olumsuzkesisim =list(set(olumsuz) & set(kelimelistesi))

    olumlukesisim = len(olumlukesisim)
    olumsuzkesisim = len(olumsuzkesisim)

    if olumlukesisim> olumsuzkesisim:
        print("Cümle olumludur.")

    elif olumlukesisim == olumsuzkesisim:
        print("Cümle hakkında bir bilgin yok.")

    else:
        print("Cümle olumsuzdur.")

duyguanalizi()

```

Örnek ekran çıktılarımıza bakalım.

```

Duygu Analiz Programı
Bir cümle girin: muhteşem bir gün
Cümle olumludur.
Bir cümle girin: berbat bir gün
Cümle olumsuzdur.
Bir cümle girin: bugün ankaradayım
Cümle hakkında bir bilgin yok.
Bir cümle girin: █

```

Görüldüğü gibi içinde “muhteşem” kelimesi geçen cümle için olumlu, “berbat” kelimesi geçen cümle için olumsuz şeklinde geri dönüş yaptı. Herhangi olumlu veya olumsuz bir kelime barındırmayan cümle için ise “Cümle hakkında bir bilgin yok.” cevabını verdi.

Kodumuza tekrar dönersek, öncelikle bir fonksiyon olarak bu işlemi tanımladığımızı dikkat edelim. Bu bir mecburiyet değil fakat daha derli toplu çalışmak açısından önemli. Olumlu kelimeler ve olumsuz kelimeler için örnek olması amacıyla 3'er kelimedenden oluşan bir liste koyduk. Bu listedeki kelime sayısını istediğimiz kadar artırabiliriz. Hatta kelime listelerimizi

metin belgelerinde tutarak bu belgeleri de programa liste olarak okutabiliriz. Unutmayalım ki kelime sayısı ne kadar artarsa programın hassaslığı o kadar artacaktır. Ardından set fonksiyonu ve && ifadesiyle iki kesişim kümesi hesapladık. len fonksiyonuyla kesişim kümelerinin büyüklüğünü bulduk. Örneğin “Bugün harika bir gün, muhteşem hissediyorum. O berbat hisler içimden gitti.” şeklinde bir metin geldiğinde olumlu kesişimimiz iki kelime olacak. “muhteşem ve harika” olumsuz kesişim ise “berbat” kelimesi. Olumlu sayısı olumsuzdan büyük olduğu için ekranda “Bu cümle olumludur” yazacak.

Bu şekilde örnek cümlelerimizi yazdırdık. Şöyle de düşünebiliriz. Bir okula gönderilen e-postaların istek mi , şikayet mi olduğunu da benzer bir çalışmayla bulabiliriz. Bunun için istek durumlarında ve şikayet durumlarında kullanılan kelimeleri liste olarak programımıza öğretmemiz yeterli olacaktır. Elbette bu yöntem her zaman %100 çalışır diyemeyiz. Ama ne kadar kaliteli bir kelime listemiz olursa oran o kadar artacaktır.

Palindrom Yakalayıcı

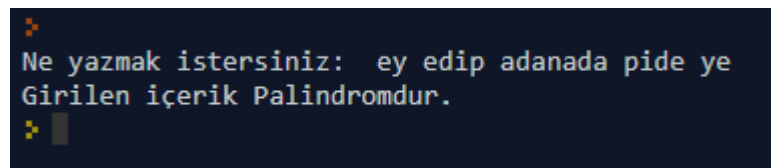
Siz hiç “Ey edip Adanada pide ye” lafını duydunuz mu ? “Buğra abi o da ne demek?” diyenler var görüyorum. Bu cümlelerin özelliği tersten söylenince de aynısı olması. Deneyin bakalım. Oldu değil mi ? İşte bu tarz kelimeler,cümleler, sayılara Palindrom deniliyor. Örneğin 363 sayısı veya Ada kelimesi. Şimdi sorumuza geçelim.

Kullanıcı tarafından girilen değişkenin Palindrom olup olmadığını bulan bir program yazın.

```
metin = input ("Ne yazmak istersiniz: ")
metin = metin.casefold()
tersmetin = reversed(metin)

if list(metin) == list(tersmetin):
    print("Girilen içerik Palindromdur.")
else:
    print("Girilen içerik Palindrom değildir.")
```

Ekran çıktımıza bakalım.



```
Ne yazmak istersiniz: ey edip adanada pide ye
Girilen içerik Palindromdur.
```

Kodumuza tekrar dönecek olursak öncelikle metin isminde bir değişkeni kullanıcıdan aldık. Ardından casefold() metodunu çalıştırdık. Bu şekilde büyük harf, küçük harfi farkını önemsiz hale getirdik. Reversed fonksiyonu ile metnimizi tersten yazıp bunu tersmetin isimli değişkene eşitledik.

Ardından bir kontrol yapısı çalıştırdık. Eğer metin, ters metine eşit olursa “Girilen içerik Palindromdur.” değilse “Girilen içerik Palindrom değildir.” şeklinde sonuç döndü.

Okuyucuya not:

Kitabın 0.7 versiyonuna göz attınız. Yeni versiyonlar ve eklenmesini istediğiniz bölümler için lütfen iletişim kurun.

Buğra AYAN

bugra.ayan@yahoo.com
<http://www.facebook.com/bugraayan>
<http://www.twitter.com/bugraayan>
<http://www.instagram.com/bugraayan>