Python Eğitim Kitabı

Volkan Taşcı

Volkan Taşcı

volkantasci.com 

youtube.com/volkantasci 

instagram.com/angelryuk/ 

linkedin.com/in/volkantasci/ 

github.com/volkantasci 

proprogramci@outlook.com 

İçindekiler

KISIM 2 MACERA BAŞLIYOR

KISIM 8 HERŞEY BİR OBJEDİR

1

Python Hakkında

Bu Bölümde

Python, son yılların popüler programlama dilleri arasında olmasından dolayı ismini sık sık duymuş olmalısınız. Aksi durumda Python hakkındaki merakınızı gidermek için bu kitabı okuyor olmazdınız.

O halde bu bölümde, popüler bir programlama dili olan Python nedir? ve ne değildir? gibi sorulara yanıtlar verelim.

PYTHON NEDİR?

Python, **Guido van Rossum** adlı Hollanda'lı bir programcı tarafından geliştirilmiştir. C, C++ gibi dillerin aksine derlenebilir değil yorumlanabilir bir dildir. (Yorumlama ve derleme konularına ileride değineceğiz) Bununla birlikte Python, nesne yönelimli, yüksek seviyeli ve modüler bir programlama dilidir. C gibi dillere nazaran fonksiyonların süslü parantezleri yoktur, programcıyı parantez karmaşasından kurtarır. Parantezler yerine Python, girinti mekanizmasını geliştirmiştir.

Python, diğer programlama dilleri gibi bilgisayarlara, telefonlara hatta elektronik kartlara istediğimiz şeyleri yapmasını söyleyebilmemiz için geliştirilmiş bir dildir. Diğer dillere göre öğrenilmesi çok daha kolay ve programlamaya başlangıç için ideal diyebiliriz.

C ve türevi diller başlangıçta hiçbir fonksiyonu tanımazlar. Herhangi bir fonksiyonu kullanmak için ilgili kütüphaneyi programa dahil etmemiz gerekir. Ekrana basit bir çıktı göndermek için bile bir kütüphaneyi dahil etmeli, `main` fonksiyonu yazmalı ve içerisinde ekrana çıktı bastırma fonksiyonunu çağırmalıyız. Tüm bunları yaparken de kodlarımızın sonuna noktalı virgül koymak zorunda kalıyoruz.

Ancak Python, başlangıçta pek çok fonksiyonu tanır. Yani pek çok işlem için bir kütüphaneyi (Python' da buna modül diyoruz) programa dahil etmemiz gerekmiyor. C dilinde beş satır kod ile ancak ekrana çıktı gönderebilirken, Python dilinde bunu yalnızca bir satırda yapabiliyoruz. Ayrıca kodlarımızın sonuna noktalı virgül koymak gibi bir zorunluluğumuzda yok. Binlerce kod içerisinde bir satırın sonuna noktalı virgül koymayı unuttukları için kriz geçiren programcılar olduğu bilinir. Python sayesinde böyle bir hata yapmamız olanaksızlaşıyor.

Tüm bunlara ilaveten Python, günümüzde elektronik alanlarda oldukça kullanılmakta. Buna en basit iki örnek **PyBoard** ve **Raspberry Pi** projeleridir. PyBoard, Python dili ile kodlanabilen elektronik bir karttır. Raspberry Pi ise küçük bir bilgisayardır ve GPIO pinleri yine Python ile kontrol edilebilmektedir. **Örneğin;** Raspberry Pi ve Python kullanılarak 3D Printer cihazları üretilebilmektedir.

Modüler bir yapısı olduğunu söylemiştik. Yukarıda bahsettiğimiz elektronik projeler için kullanılan modüller gibi hemen her iş için Python'da modül vardır. **Python;** Django, Flask gibi framework'ler ile web geliştirme konularında da çokça kullanılmaktadır.

Python dilinin piyasada bulunan iki sürümü vardır. **Python 2x** ve **Python 3x**. Python 2 sürümü ile artık gelişme sağlanamadığından Python 3 geliştirilmiş ve zamanla piyasadaki tek sürüm olma yolunda ilerlemektedir. Ancak halen Python 2 sürümünü kullanan programlar olduğundan, bazı modüllerin henüz Python 3'e geçişi tam olarak sağlanmadığından Python 2'nin kullanımı da devam etmektedir. Biz bu kitapta Python 3.7 sürümünü kullanacağız.

Neden Python Dili?

Python'un kolay okunabilen ve yine kolayca yazılabilen bir sözdizimi olduğundan bahsetmiştik. Hatta bu sebeple de kolay öğrenilebilir olduğunu, başlangıç için ideal bir dil olduğunu belirttik. İşte bu sebepler pek çok büyük firmaların da bu dili kullanmalarına ve geliştirmelerine sebep olmuştur. Aynı şekilde büyük firmaların Python kullanıyor olması, Python bilen programcıların da iş fırsatlarını daha kolay elde etmesini sağlar. Python' un kendi özellikleri haricinde yukarıda belirtmiş olduğumuz sebep bile Python dilinin neden öğrenilmesi gerektiğine basit bir örnektir.

Derlenebilir ve Yorumlanabilir Ne Demektir?

C ve türevi dilleri derlenebilir dillere örnek olarak gösterebiliriz. Bu diller ile uygulama geliştirirken uygulamanın derlenme aşaması vardır. Yani kod yazım işlemi bittikten sonra bu kodları bilgisayarımızın anlayacağı şekle çevirmeye ihtiyaç duyarız. Bunu yapabilmek için de kullandığımız dile uygun derleyiciyi bilgisayarımıza kurmamız gerekir.

Bilgisayarlar, bizim yazdığımız kodları direkt olarak anlayamazlar. Bilgisayarların anlayabildiği tek dil vardır o da makine dili. Nedir bu makine dili biraz da onu açıklayalım.

Günlük hayatta kullandığımız sayılar gibi başka sayılar da vardır aslında. **Örneğin;** bizim sayı sistemimizde en yüksek rakam dokuzdur. Ancak öyle sayı sistemleri vardır ki en yüksek rakamı bizim sayı sistemimizde on beş olarak gösterilir. Toplamda da on altı rakamı vardır. **Örneğin;** A, B, C rakamlarının bizim sayı sistemimizdeki karşılığı 10, 11, 12'dir. İşte bu sayı sistemine on altılık sayı sistemi (Hexadecimal) denir ve bilgisayarların yani makinelerin de kullandığı dil budur. Derleyicilerin yaptığı iş bizim kodlarımızı alıp on altılık sayı sistemine dönüştürmektir. Eğer kodlarda bir hata varsa derleme işlemi hata verir ve programı düzenlemek zorunda kalırız.

Yorumlama işlemi ise bu derleme olayını tek seferde tüm kodlara uygulamaz. Yani kodların onuncu satırında hata varsa da program ilk etapta çalışır. Programın akışı onuncu satıra ulaştığında program hata verir ve kapanır. Ancak baştaki dokuz kod işletilir.

Programlama Dili Nedir?

Herşeyi temelden alıp öğrenmeyi hedefliyorsak ilk bilmemiz gereken konu da budur. Programlama dili tam olarak nedir? Yalnız oyun mu yazılır programlama dilleri ile, sadece telefonlara uygulama mı yapılır?

Sürekli bir şehir efsanesi dolanır dillerde, "programlar 1 ve 0 ile yazılıyormuş!" Elbette ki **HAYIR**. Özellikle bu kitapta öğreneceğimiz Python programlama dili, insan diline çok yakın bir dildir. Python programlama dilinde kullanılan terimlerin İngilizce kelimelerden oluşması öğrenmesini ve yazmasını kolaylaştırır.

3

Komut Satırı Bilgisi

Bu Bölümde

Bu bölümde etkileşimli kabuk ile Windows'un ya da Linux'un komut satırı arasındaki farka değineceğiz. İlaveten Windows'un komut satırını temel düzeyde (işlerimizi yapabileceğimiz kadar) öğreneceğiz.

Etkileşimli kabukta Python kodları yazabiliyorduk. Buna ilaveten hesap makinesi gibi çalıştığını da söylemiştik.

Örneğin;

```
>>> 3 + 5
8
>>>
```

Etkileşimli kabukta yaptığımız bu örnek hesap makinesi gibi de çalıştığının bir kanıtı. Ancak Windows ya da Linux'un komut satırı ile etkileşimli kabuğu birbirine karıştırmamak gerek. Etkileşimli kabukta Python kodları çalışırken işletim sisteminin komut satırında yalnızca sistem komutları çalışır. Yukarıda yaptığımız örneği Windows'un komut isteminde ya da Linux'un terminalinde yapamayız. Yukarıdaki örnek bir etkileşimli kabuk örneğidir. Bunu yapabilmek için terminalden ya da komut isteminden **Python Etkileşimli Kabuğu** çalıştırmamız gerek. Daha önceki bölümde yaptığımız gibi komut satırına **python** yazarak bu işlemi yapıyoruz. Buna alternatif olarak IDLE programı üzerinden de etkileşimli kabuğa ulaşabildiğimizi hatırlamakta fayda var.

Yine önceki bölümlerde bu siyah ekranı epey kullanacağımızdan söz etmiştik. Bir programcı olarak Python bilgisinden önce komut satırı bilgisine ihtiyaç var. Bu siyah ekranı, en azından işlerimizi yapabilecek kadar öğrenmeliyiz.

Bilgisayarda yaptığımız işlemler aslında komut satırına birer kod göndermek oluyor. Açalım bu ifadeyi. Her bir fare tıklaması, her bir tuşa basmak, kopyalamak, yapıştırmak vs. tüm işlemlerimiz komut satırında yapılıyor aslında. Arka planda gerçekleştiği için göremiyoruz. Yeri geldiğinde biz de Python kodlarımız ile komut satırına bazı kodlar göndereceğiz.

Madem her işlem aslında bu siyah ekranda gerçekleşiyor o halde bazı şeylerin nasıl olduğuna bakmakta fayda var.

WINDOWS KOMUT SATIRI

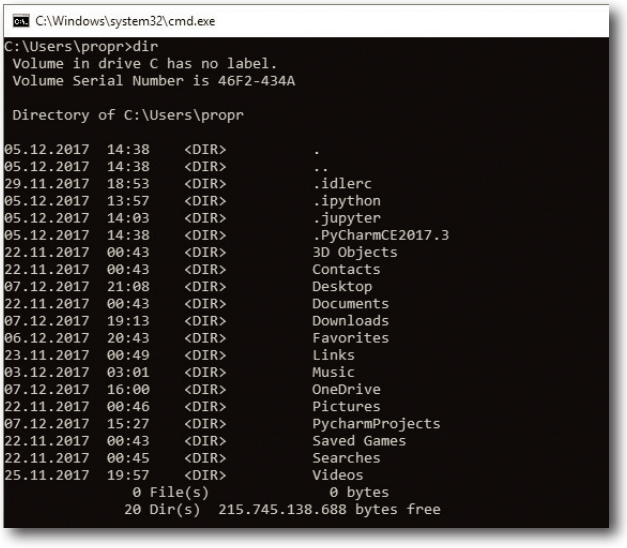
Windows kullanıyorsanız ve bilgisayarla haşır neşir değilseniz komut satırına pek ihtiyaç duymamış olabilirsiniz. Ancak programlamadan bahsediyorsak bu siyah kutucuklarda ne oluyor ne bitiyor öğrenmek zorundayız. Biz bu kitapta işimize yaracak olan kısımlara değineceğiz. Ancak siz programcılık maceranızda komut satırı ile ilgili çok şey öğreneceksiniz.

dir Komutu

Bazen bir dosyaya ulaşabilmek için bulunduğumuz dizinden başka dizine geçeriz. **Örneğin;** Masaüstü klasörünün bulunduğu adres **C:\Users\propr\Desktop** adresidir. Benim kullanıcı adım **propr** olduğu için **Desktop** klasöründen önce gelen dizin bu. Sizin kullanıcı adınız da o kısımda bulunacak. Biz Masaüstünde bir klasörü açtığımızda aslında yaptığımız işlem yukarıdaki adres altında bulunan bir klasörü açmak oluyor.

Örneğin;

C:\Users\propr\Desktop\YeniKlasör adresine ilerlemiş olalım. Bunu yapabilmek için önce Masaüstünde bulunan içeriğe bakıyoruz. **YeniKlasör** adlı klasörü önce görüyoruz sonra da o klasöre ilerliyoruz. İşte `dir` komutunun yaptığı iş tam olarak bu. Herhangi bir dizinin içeriğini görüntülemek. Komut İstemini açıp şu `dir` yazalım. Aşağıdakine benzer bir çıktı almış olmalısınız;



Grafiksel olarak da kullanıcı klasörünüze gidip kontrol edebilirsiniz.

Biz bir kod yazmadan ekranda beliren `C:\Users\propr>` kısmı bize hangi dizinde çalıştığımızı söylüyor. Yani biz dosyaları listelemek için `dir` komutunu kullandığımızda `C:\Users\propr>` dizini altında bulunan dosya ve klasörleri listeliyor. Listelenenler arasında nokta ve iki nokta olarak isimlendirilmiş ve klasör olarak da belirtilmiş iki klasör var. Bunlara bir sonraki komutta değineceğiz.

NOT Dosya ve klasörler listelendiğinde üçüncü sütunda `DIR` diye belirtilenler klasördür.

4

Programlamaya Giriş

Bu Bölümde

Bu bölümde, Python dilinde bulunan temel veri tiplerinden (Integer, Float, String) detaylıca bahsedeceğiz.

Bu verileri saklayan değişkenlerden, bu değişkenlerin nasıl tanımlandığından ve nasıl çağrıldığından örnekler vererek konuşacağız. Çok satırlı programlar yazabilmek için Python dosyalarının nasıl oluşturulduğunu göreceğiz. Son olarak da bu verileri ekrana bastırabilmek için gerekli fonksiyon olan print() fonksiyonuna detaylarıyla değineceğiz.

Son olarak da bu verileri ekrana bastırabilmek için gerekli fonksiyon olan `print()` fonksiyonuna detaylarıyla değineceğiz.

TEMEL VERİ TİPLERİ

Veri tiplerini açıklamadan önce genel bir tanım yapalım. Bilgisayarları birer aptal makine olarak düşünmeliyiz. Onlardan bir şey isterken her şeyi ama her şeyi söylemeliyiz. Eğer iki sayıyı toplamasını istiyorsak önce o sayıların matematiksel bir ifade olduğunu belirtmeliyiz. Hatta öyle ki virgüllü sayı ile tamsayıyı bile farklı farklı söylemeliyiz. Yani bilgisayarlar için 5 ile 5.0 sayıları arasında fark vardır. Veya sayı olan 5 ile yazı olan '5' arasında fark vardır. Bilgisayarlara işlem yapmasını söylerken bunları hep bilirtmeliyiz.

Yukarıdaki işlemlerin çoğunu Python, bizim yerimize yapıyor olsada bazı yerlerde bizim belirtmemiz gerekiyor. **Örneğin;** iki sayıyı topla dediğimizde Python bunların sayı olduğunu anlıyor ve bilgisayarımıza ona göre bir cevap gönderiyor. Ancak bazen öyle durumlar oluyor ki Python'a **yazı olan '5'i de sayı olarak kabul et** gibi şeyler söylememiz gerekiyor. Bunu da yeri geldikçe göreceğiz.

DEĞİŞKENLER

Değişkenler, kendi içlerinde bazı değerler tutarlar. **Örneğin;** 8 gibi, 6 gibi, 107 gibi… Bu değerler tamsayı olmak zorunda değil. İçlerinde 9.5, 4,2 gibi sayıları da tutarlar. Hatta ileride bazı veri tipleri öğreneceğiz ki bu veri tipleri içlerinde değişken tutan değişkenleri barındıracaklar. Aynı şekilde bu değerlerin sayı olması gibi bir zorunluluk da yoktur. İçlerinde **Merhaba Dünya** gibi bir yazıyı da tutabilirler. İsimlerinden de anlaşıldığı gibi bunlar program içerisinde değişkendirler. Programın akışına göre sürekli farklı değerler alabilirler.

Bir değişken oluşturmak oldukça kolaydır. Aşağıdaki örneği inceleyelim;

```
>>> a = 56
>>> b = 32
```

Değişken oluştururken kullandığımız operatör, eşittir operatörü. `a = 56` ifadesi ile Python'a diyoruz ki **a dediğim yere artık 56 yazacaksın!**

Aynı şekilde `b = 32` ifadesi ile de `b gördüğün yere 32 yazacaksın Python!` diyoruz. Bundan sonra programda `a` ve `b` yazdığımız yerlerde yukarıdaki değerler geçerli olacaktır.

Değişkeni oluşturduktan sonra onu kullanmak istediğimiz her yere yazabiliriz. Bundan sonra nereye `a` yazarsam orada sayısal olarak `56` değeri verecektir. Ve aynı şekilde nereye `b` yazarsam orada sayısal olarak `32` değeri olacaktır.

Yorumlayıcı ekranının bir hesap makinesi gibi de çalıştığını söylemiştik. O halde `a + b` yazarsak bakalım ne olacak.

```
>>> a = 56
>>> b = 32
>>> a + b
88
>>>
```

Görüldüğü gibi a değişkeninin değeri ile b değişkeninin değeri toplandı ve ekrana basıldı. Değişkenlerin içlerinde yazıları da barındırabildiğinden bahsetmiştik, bunu da deneyelim;

```
>>> yazi = "Merhaba Dünya"
>>> yazi
'Merhaba Dünya'
>>>
```

Yukarıda yaptığımız işlemleri anlatayım ki öğrendiklerimiz pekişmiş olsun.

İlk olarak dikkatimizi değişkenin ismi çekiyor. Daha önceden a ve b gibi tek harften oluşan isimler verirken bu defa bir kelime olarak isim verdik. Python için bunun bir mahsuru yok, yani istediğimiz kelimeyi değişken ismi olarak verebiliriz ancak bazı kurallar var. Değişken isimleri sayı ile başlayamaz, - + * / gibi karakterlerden oluşamaz, bu karakterler gibi **nokta** (**.**) karakteri de kullanılamaz. Ancak değişken isimleri **alt çizgi** ile _ başlayabilir. Sayı ile başlayamasa da içerisinde sayı bulundurabilir. Bir diğer önemli husus da küçük ve büyük harf duyarlılığıdır. Yani sayi ile Sayi birbirinden farklı değişkenlerdir.

Bir diğer dikkati çeken husus ise yazı değil de yazi yazmış olmamız. Bunun sebebi Python hatalarından kaçınmak değil. Yani biz oraya yazı da yazmış olsaydık da gayet düzgün çalışacaktı. Ancak programlamada bir prensip olarak **İngilizce** karakterler kullanıyoruz ki klavyesinde **Türkçe** karakter bulunmayan insanlar da kodlarımız üzerinde düzenlemeler yapabilsin.

yazi değişkenine **Merhaba Dünya** diyerek bir değer atadık. Bundan sonra bir yere yazi ibaresini koyarsak orada bulunan değer **Merhaba Dünya** olacaktır.

7

Kullanıcı ile Etkileşim ve Veri Dönüşümleri

Bu Bölümde

Bu bölümde programlarımızın kullanıcı ile etkileşim halinde ilerlemesini sağlamayı öğreneceğiz. Bununla birlikte kullanıcıdan gelen girdileri farklı veri tiplerine dönüştürmeyi öğreneceğiz.

GİRİŞ

Şu ana kadar yaptığımız örnekler oldukça basit ve tek yönlülerdi. Tek yönlü diyerek tek olasılıktan bahsediyorum. Ve bu da pek güzel bir şey değil. Önceki bölümlerde uyguladığımız yöntemler gibi kısa bir örnek ile başlayalım;

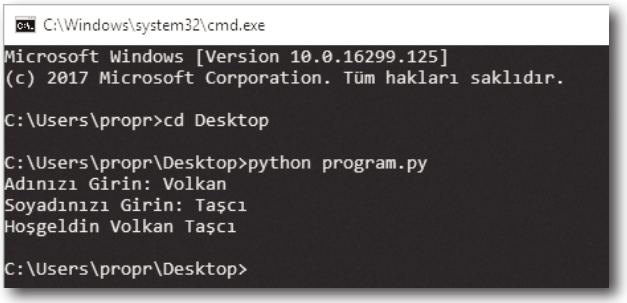
```
>>> isim = "Python"
>>> sınıf = "Programlama Dili"
>>> print(isim,sinif)
Python Programlama Dili
>>>
```

Burada yaptığımız şey direkt olarak programın kodları içerisinde değişkene değer vermekten ibaret. Ancak programımız bize adımızı, soyadımızı sorsa ve bizim gireceğimiz bilgilere göre hoş geldin mesajı yansıtsa güzel olmaz mı? Doğal olarak artık tek yönlü değil kullanıcının girdiği bilgilere göre farklı sonuçlar üreten bir program haline gelir değil mi? Bunu yapabilmek için değişkenlerimize direkt olarak değer atamıyoruz. Değişkenlerimize `input()` fonksiyonu yardımıyla kullanıcının girdiği veriyi atıyoruz. Örnek koda bakalım.

```
ad = input("Adınızı Girin: ")
soyad = input("Soyadınızı Girin: ")
print("Hoşgeldin",ad,soyad)
```

`ad` ve `soyad` değişkenlerine direkt değer atamak yerine `input()` fonksiyonunun döndürdüğü değeri atıyoruz. Fonksiyonun kullanımına değineceğiz ama önce şu **döndürmek** kavramını inceleyelim. `input()` fonksiyonunun bir değer döndürdüğünden bahsettim. Nedir bu?

Fonksiyonların her biri bir işlemi yerine getirir ve bunun sonucunu bizlere verir. Sonucu verme işlemine döndürmek diyoruz. Peki, bu sonuç nerede? Yukarıdaki örnek üzerinden anlatayım. `ad` değişkeninin karşısında bulunan ifade, fonksiyonun bize döndürdüğü değer olur. Yani hayalinizde kodların o bölümünde başka bir değer yazdığını düşünün, kullanıcının girdiği değer. **Örneğin;** `ad` değişkeninin karşısında `'Ayşe'`, `soyad` değişkeninin karşısında `'Taşcı'` yazıyormuş gibi düşünebilirsiniz. Çünkü, fonksiyon çalıştığında bizden aldığı değeri oraya yerleştirecek.



Programı çalıştırıp önce adımızı giriyoruz ve **Enter'a** basıyoruz. Ardından aynı şeyi soyadımız için yapıyoruz. Bilgilerimizi girdikten sonra Python bizi selamlıyor. Şimdi `input()` fonksiyonunu biraz inceleyelim.

input() Fonksiyonu

`input()` fonksiyonunu kullanıcıdan String veri tipinde bir veri almak için kullanıyoruz. Unutmayın, yalnızca String türünde alabiliyoruz `input()` fonksiyonu ile.

Dikkat ederseniz yukarıdaki örneğimizde fonksiyona bir argüman verdik. Bu argüman, fonksiyon çalıştığında ekranda görünecek yazıdır. Hatırlatmakta fayda var, argüman olarak `input()` fonksiyonuna yalnızca bir tane ve onu da String veri tipinde verebiliyoruz. Şimdi bir örnek daha yapalım;

Örnek Kod

```
sayi1 = input("Sayı1: ")
sayi2 = input("Sayı2: ")
print("İşlem Sonucu:", sayi1 + sayi2)
```

Çıktı

```
Sayı1: 2
Sayı2: 3
İşlem Sonucu: 23
```

Hatırlarsanız String cinsindeki verileri toplarsak Python verileri yan yana ekliyordu. Burada da bunu görüyoruz. O halde kullanıcıdan sayısal veriler alıp bunları matematiksel işlemlere tabi tutmak istersek nasıl bir yol izleyeceğiz? Veri tiplerinin birbirine dönüşümünü öğrendiğimiz zaman bu sorunun da üstesinden geleceğiz. Ancak kitabımızın **Giriş** kısmında anlattıklarımızdan belki çıkarımlar yapmışsınızdır. Orada da söylediğimiz gibi bazı durumlarda Python'a `yazı içerisindeki '5' ifadesini sayı olarak kabul et` gibi şeyler söylememiz gerekecek. İşte bu işlemin teknik tanımına veri tipleri arasında dönüşüm diyoruz.

10

Hata Yakalama

Bu Bölümde

Bu bölümde programlarımızda oluşabilen hataları düzeltme yöntemlerini ve olası kullanıcı hatalarını gidermek için gerekli olan metotları öğreneceğiz.

Bununla birlikte hata yakalama işlemlerini farklı amaçlar için kullanabileceğimizi göreceğiz.

HATA YAKALAMA NEDİR?

Programlarımızda bazı hatalar ile karşılaşmamız mümkündür. Örneğin kullanıcıdan sayı isterken kullandığımız metot hatalara açıktı. Aşağıdaki örnek kod ile kullanıcımızdan bir tamsayı girmesini istediğimizi varsayalım;

```
sayı = int(input("Tamsayı girin: "))
```

Bu kodları çalıştırıp sayı yerine bir harf girersek veya sayı ile karışık harf girersek program şöyle bir hata mesajı gösterip sonlanacaktır;

```
C:\Users\propr\Desktop\Egitim>python program.py
Tamsayı girin: a
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    sayı = int(input("Tamsayı girin: "))
ValueError: invalid literal for int() with base 10: 'a'
```

Biz, kullanıcıdan alınan karakter dizisini `int()` fonksiyonu ile bir tamsayı yapmaya çalışıyoruz. Eğer kullanıcı tamsayı girmez ise doğal olarak çöküyor programımız.

Hata yakalama metotları ile bu ve benzeri hatalardan kurtulabiliriz hatta kurtulmak ile beraber böyle bir şeyin gerçekleşmesi durumunda neler yapılması gerektiğini ayarlayabiliriz. Yukarıda bulunan hata mesajı biz programcılar için bir şeyler ifade etse de kullanıcılar için tamamen anlamsız yazılar olacaktır. Eğer böyle durumlarda yine de programın sonlanmasını ama kullanıcıya kullanıcının dilinden mesaj görünmesini istersek bunu da ayarlayabiliriz.

HATA ÇEŞİTLERİ

Hataları birkaç sınıfa ayırıp inceleyebiliriz. Çünkü program hataları sürekli aynı şekillerde karşımıza çıkmaz.

PROGRAMCI HATASI

Programcı hataları program kullanıcıya ulaşmadan fark edilip çözülen sorunlardır. Basitçe sözdizimi hatları diyebiliriz. Örneğin C dilinde kodlama yapan birisi için noktalı virgül koymayı unutmak gibi bir şeydir. Python dilinde ise girinti açılmasına rağmen koşulun sonuna iki nokta koymamak bir programcı hatasıdır. Ancak daha öncede söylediğimiz gibi bu hatalar program kullanıcıya ulaşmadan çözülen hatalardır. Çünkü bu hataları çözmeden program zaten hata verip sonlanacaktır.

Program Hatası – Bug

Program hatalarında program sorun çıkarmadan çalışır yani durduk yerde sonlanmaz. Ancak programın üretmiş olduğu sonuçlar doğru değildir. Bu hatayı fark etmek zor olabilir ve fark edilmeden kullanıcıya sunulabilir. Ancak kullanıcıların geri dönütleri sayesinde bu hatalar da çözülebilmektedir. Örneğin bir hesap makinesi programı yazıyorsunuz ve kullanıcıdan sayılar alıyorsunuz. Eğer bu sayıları sayısal değerlere dönüştürmeden toplama işlemi yaptırırsanız program sonlanmaz. Ama programınız hatalı sonuç üretir. Karakter dizilerinin toplama işleminde yan yana yazıldıklarını biliyorsunuz. **Örneğin;** 5 ile 2 sayısını toplamaya kalktığımızda 52 sayısını üretecektir. Daha doğrusu 52 çıktısını basacaktır ekrana. Bu da hatalı sonuç demektir.

İstisnai Durumlar

Programımızın bazen çalışıp bazen ise çalışmaması gibi durumlara verilen isimdir istisnai hata. Programımız her ne kadar doğru olsa da bazen sisteme göre, kullanıcıya göre çalışmada sorun yaşayabilir. Örnek olarak şöyle bir kod çalıştıralım;

```
import os

userFolder = os.path.expanduser('~')
os.chdir(userFolder + '\\' + 'Desktop')

with open('dosya.txt'):
    print('Dosya Mevcut')
```

Bu kodu çalıştırdığınızda kiminizde hata verecek kiminizde ise çalışacaktır. Belki de hepinizde çalışacak belki de hepinizde hata verecektir. Çünkü bu kodların yaptığı iş sizin Masaüstünüzde bulunan `'dosya.txt'` dosyasını açmaya çalışmak. Yani o dosyayı açabilmesi için sizin Masaüstünüzde `'dosya.txt'` dosyasının bulunması gerek. Eğer yoksa hata verecektir. Eğer hata alıyorsanız bir de yeni dosya açıp adını `'dosya.txt'` yapıp yeniden çalıştırın kodu. Bu defa hata vermediğini göreceksiniz. Veya sisteminiz Windows değilse yine hata verecektir. Çünkü dosyaya giden yolu Windows sisteme göre yazdık.

İstisnai durumlara bir örnek daha verelim. Hesap makinesi yazdığımızı düşünelim. Ama matematiğe aykırı bir durum vardır aslında hesap makinelerinde. Biz programa her işlemi yaptırmaya çalışırsak bir durumda hata verir. Bu durum sıfıra bölme durumudur. Bir sayının sıfıra bölünmesi tanımsızdır. Programımıza bu istisnai durumun çözümünü yazmazsak matematiği test etmek isteyen bir kullanıcı bu işlemi deneyebilir ve program sonlanabilir.

Tüm bu üç hata şeklinin de çözümleri var. İşte bu bölümde bu hataları nasıl engelleyeceğimizden bahsedeceğiz.

12

Karakter Dizilerini Biçimlendirmek

Bu Bölümde

Karakter dizilerinin büyük öneme sahip olduğunu daha önce de söylemiştik. Ne kadar kolay değişiklik yapabiliyorsak bu veri tipinde o kadar kolaylaşır verilerin birbirine dönüşmesi. Bu bölümde de karakter dizilerini biçimlendirerek kolayca değişiklik yapmayı göreceğiz.

FORMAT METODU

Önceki bölümde metotlar konusunu işledik fakat bu metottan bahsetmedik. Karakter dizilerini biçimlendirme görevini üstlenen metodumuz bu. Peki, biçimlendirmek derken ne demek istiyorum?

Örnek koda bakalım;

```
isim = 'Python'
print("Merhaba Dünya, benim adım",isim)
```

Bu kod gayet güzel iş görür ancak daha karmaşık bir metin olsaydı değişkenleri metin içerisinde nasıl kullanabilirdik? Yine bir örnek kod ile açıklayalım;

```
metin = """Python, nesne yönelimli, yorumlamalı, birimsel (modüler)
ve etkileşimli yüksek seviyeli bir programlama dilidir.

Girintilere dayalı basit sözdizimi, dilin öğrenilmesini ve akılda kalmasını
kolaylaştırır.
Bu da ona söz diziminin ayrıntıları ile vakit yitirmeden programlama yapılmaya
başlanabilen bir dil olma özelliği kazandırır.
"""
```

Bu uzunlukta ve çok satırdan oluşmuş bir metin içerisine bir şeyler eklemek istersek yukarıdaki yöntemle yapmamız çok zor olur. Önce metni ikiye bölüp araya virgül ile bir değişken koyup ardından metnin kalanını eklememiz gerekir. Bu oldukça meşakkatli bir yol. Bunun yerine format metodu çok kullanışlı olacaktır. Örnek olarak şu koda bakalım;

```
metin = """Python, nesne yönelimli, yorumlamalı, birimsel (modüler)
ve etkileşimli yüksek seviyeli bir programlama dilidir. {}

Girintilere dayalı basit sözdizimi, dilin öğrenilmesini ve akılda kalmasını
kolaylaştırır.
Bu da ona söz diziminin ayrıntıları ile vakit yitirmeden programlama yapılmaya
başlanabilen bir dil olma özelliği kazandırır.
"""
print(metin.format('Bilimsel çalışmalarda kullanılır.'))
```

Metin içerisindeki süslü parantezler dikkatinizi çekmiş olmalı. `Format` metodunun görevi; sırasıyla bu süslü parantezlere gelecek olan verileri almaktır. Biz yukarıda yalnızca bir adet veri aldık çünkü yalnızca bir adet süslü parantezimiz vardı.

Şimdi bu sayıyı artıralım;

```
isim = 'Volkan'
soyisim = 'Taşcı'
print('Benim adım {} {}'.format(isim, soyisim))
```

Şu örnekle de hangi verileri alabileceğimizi görelim;

```
x = 5
y = 6
print('{} + {} = {}'.format(x, y, x+y))
```

Çıktımızı kontrol edelim;

```
5 + 6 = 11
```

Demek ki yalnızca karakter dizisi olarak değil sayısal verileri de alabiliyormuşuz.

Format metodunu kullanmak için sıralı değerler vermek şart değil. Yani ilk açılmış olan süslü parantezin değerini en son yazabiliriz. Bunun için şöyle bir yöntem izliyoruz;

```
x = 5
y = 6
print('{a} + {b} = {sonuc}'.format(b = y, a = x, sonuc = x+y))
```

Biraz önce yazdığımız kodlarla aynı çıktıyı alıyoruz. Yalnızca farklı sıra ile yazdık.

Format metodunun işlevi bu kadar basit değil. Daha pek çok marifeti var. Şimdi sırasıyla oları görelim.

Belli Bir Alanda Sağa veya Sola Yaslayarak Biçimlendirme

Yazının, önceden ayrılmış belli bir alana yazılmasını sağlamak amacıyla kullanılır. Ayrılan alandan küçük olsa dahi yazımız, sonraki yazılacak olan karakter alandan sonra bastırılacaktır.

```
string = "|{:<15}|".format("Volkan")
print(string)
```

14

Demet Veri Tipi

Bu Bölümde

Bu bölümde ileri seviye veri yapılarından biri olan demet (tuple) veri tipini inceleyeceğiz.

Liste veri yapısından farkı ve benzerliği konusunu öğrenip nerelerde kullanılacağına dair örnekler yapacağız.

DEMET VERİ YAPISI NEDİR?

Demetler, tıpkı liste ve karakter dizileri gibi sıralı veri yapısıdır. Sıralı veri yapılarının sahip olduğu ortak özelliklere sahiptir. Aslında en özet şekliyle, sıralı veri yapılarının yalnızca ortak özelliklerine sahiptir. Yani sıralıdır. Bir indeks değerine sahiptir. Ek olarak liste gibi tüm veri tiplerini içerisinde barındırabilir.

Listelerden farklı olarak değiştirilemeyen bir veri yapısıdır. Ancak daha önceden değiştirilemeyen veri yapısı hakkında şöyle bir kanıya varmıştık: değişiklik yapıldığında hafızadaki farklı bir alan işaret edilir, yani yeniden tanımlama yapılmıştır. Demetlerde durum daha farklı, demetler hiç değiştirilemez. Yani değiştirmek gibi bir işlem yapılmaya kalkışılırsa hata alınır.

Tanımlama işleminde veri tiplerinin belirteçleri olduğunu görmüştük. Karakter dizileri için tırnak işaretleri, listeler için köşeli parantezler gibi. Demetlerin de belirteçleri vardır ve normal parantezlerdir.

DEMETLERİN KULLANIMI

Demetleri tanımlarken normal parantezlerden faydalanabiliyoruz. Örneğin üç öğeli bir demet tanımlayalım.

```
demet = ('Ali', 'Veli', 'Kırk Dokuz Elli')
```

Demet tanımlamak da liste gibi oldukça kolay.

Boş liste tanımlaması yaparken şöyle bir kod yazabiliyorduk hatırlarsanız;

```
liste = [] # liste = list() şeklinde de yazabiliriz
```

Burada listelerin belirteçleri kullanılarak boş bir liste tanımlaması yapılmış. Aynı işlemi demetler üzerinde uygulayarak boş bir demet oluşturabiliriz. Bunu tahmin etmesi zaten zor değil, biliyorum. Ancak bir başka veri tipinde böyle bir şey yapamayacağız da o yüzden her bir veri yapısı için boş biçimde oluşturmayı gösteriyorum.

Örnek kod

```
demet = ()
print(type(demet))
```

Çıktı

```
<class 'tuple'>
```

tuple, yani demet olduğunu söylüyor. O halde oluşturma işlemi başarılı.

Sıralı veri yapısı olması nedeniyle sıralanmış verilere ihtiyaç duyulan her yerde kullanılabilirler. Ek olarak değiştirilmesi istenmeyen sıralı verilerde kullanılmalıdırlar. Çünkü demetleri değiştiremezsiniz. Örneğin; kütüphane sisteminde kütüphane bir demet olamaz. Ancak liste olabilir. Fakat kitaplar birer demet olabilirler. Kitabın ve yazarın adını barındıran bir demet oluşturabiliriz.

```
kitap1 = ('Ahmed Arif', 'Hasretinden Prangalar Eskittim')
```

Demetleri değiştiremiyoruz ancak demetler içerisinde değiştirilebilir veri yapılarını barındırabiliyoruz. Örneğin demet içerisinde iki tane liste oluşturabiliriz.

```
demet = (['Python', 'Ruby', 'Perl'], ['C++', 'C', 'C#'])
```

Bu demet içerisinde bulunan listelerde değişiklik yapmamız mümkün mü?

Örnek kod

```
demet = (['Python', 'Ruby', 'Perl'], ['C++', 'C', 'C#'])
demet[0].append('Lisp')
print(demet)
```

Çıktı

```
(['Python', 'Ruby', 'Perl', 'Lisp'], ['C++', 'C', 'C#'])
```

Anlaşılıyor ki demetler içerisinde bir liste varsa bu listeye erişip değişiklik yapmamız mümkün. Demetler değiştirilemeyen veri yapılarından ise ekleme çıkarma gibi metotları da yoktur. O halde hangi metotlara sahip diye bir göz atalım.

Örnek kod

```
demet = (['Python', 'Ruby', 'Perl'], ['C++', 'C', 'C#'])
for i in dir(demet):
   if "__" not in i:
       print(i)
```

Çıktı

```
count
index
```

Yalnızca iki metoda sahip. Listelerden hatırladığımız bu metotları bir de demet üzerinde test edelim.

DEMETLERİN METOTLARI

Metotlar aracılığı ile tıpkı listeleri kullandığımız gibi demetleri de daha efektif kullanabiliriz. Demetler değiştirilemeyen türde oldukları için yalnızca iki metoda sahipler ve bu metotlar demet içindeki öğelerin indeks değerlerini ve kaç adet olduklarını belirtir.

INDEX() METODU

Demet içerisindeki bir öğenin kaçıncı indekste yer aldığını döndüren metottur. Bir örnek ile açıklayalım.

Örnek kod

```
demet = ('A', 'B', 'C', 'D', 'E')
print(demet.index('C'))
```

Çıktı

```
2
```

'C' karakterinin ikinci indekste bulunduğu bilgisine ulaşmış olduk. index() metoduna zaten sıralı veri tipleri olan karakter dizileri ve listelerde de değindik. Bu sebeple üzerinde çok durmayacağım.

COUNT() METODU

Yine daha önceden incelediğimiz bir metot olduğu için üzerinde pek durmadan bir örnek ile açıklayalım.

Örnek kod

```
demet = (22,56,12,78,45,38,29,80,34,13,12,45,677,589,456,123,12)
print(demet.count(12))
```

Demet içerisindeki sayılar arasında kaç adet 12 sayısının bulunduğunu sorguladık.

Çıktı

```
3
```

Anlaşıldığı üzere demet verimizin içerisinde toplamda üç kez 12 sayısı yer almış.

16

Küme Veri Tipi

Bu Bölümde

Bu bölümde Python'da bulunan küme veri tipini öğrenip hangi amaçlarla kullanıldığına değineceğiz.

Ayrıca örnekler yaparak küme veri tipinin kullanımını pekiştireceğiz.

KÜME VERİ YAPISI NEDİR?

Matematik derslerinizden hatırlayabileceğiniz kümeler ile tamamen aynı işlevlere sahip bir veri yapısıdır. Örnek verecek olursak kümelerin kesişimlerini veya beileşimlerini sorgulayabiliriz. Tıpkı sözlükler gibi sırasız veri tiplerindendir. Yani kümelerin öğelerine indeks değeri ile ulaşamayız çünkü indeks değerleri yoktur.

KÜMELERİN KULLANIMI

Kümeler hangi alanlarda kullanılır? Bu soruya cevap olarak epey argüman üretilebilir ancak her birinin açıklamasını eklemek gereksiz bir zahmet olur. Bunun yerine kümelerin bazı özelliklerinden yola çıkarak hangi alanlarda kullanılabileceğini söylemek daha doğru olur. Kümeler içerisine bir öğe ancak bir defa eklenebilir. Yani bir küme iki aynı elemana sahip olamaz. Bu durumda kümelerin kullanılabileceği bir yol şu olabilirdi: Bir roman içerisindeki tüm kelimeleri bir kümeye dönüştürürdük ve tüm kelimelerden yalnızca bir kez eklenebileceğinden dolayı kümenin öğe sayısı aslında roman içerisinde kaç farklı kelimenin kullanıldığının bilgisidir.

Kümelerin belirteçleri yoktur. Ancak sözlüklerin belirteçleri olan süslü parantezler ile de tanımlanabilirler. Peki, sözlükler ile kümeleri nasıl ayırt ediyoruz?

Sözlükler, anahtar-değer ilişkisi ile tanımlanıyorlar. Yani ikililer arasında iki nokta işareti var. Ancak kümelerde ise öğeler iki nokta olmaksızın listelerdeki gibi yalnızca virgül ile birbirinden ayrılıyor. Bu sayede Python, süslü parantezler ile gösterilen verinin sözlük mü ya da küme mi olduğunu anlıyor.

Boş bir küme tanımlamak istersek aşağıdaki örnekte olduğu gibi `set()` fonksiyonundan faydalanmamız gerekir. Bunun sebebi yukarıda yazmış olduğumuz belirteç yoksunluğundandır. Eğer süslü parantez kullanırsak Python bu veriyi sözlük olarak algılayacaktır.

```
bos_kume = set()
```

Aşağıdaki örnekte görüldüğü biçimde **süslü parantezler** kullanılarak boş bir küme yaratılmaya çalışılırsa sözlük olarak algılanıyor.

Örnek kod

```
bos_kume = {}
print(type(bos_kume))
```

Çıktı

```
<class 'dict'>
```

Çıktı, söylediklerimizi doğruluyor. Eğer içerisinde öğe bulunan bir küme tanımlamak istersek süslü parantezleri kullanabiliriz ve iki nokta kullanmadığımız için Python bunu sözlük olarak algılamaz.

```
kume = {14, 22, 'alfa', 'beta', 45}
```

Yukarıdaki koda alternatif olarak şöyle de yapılabilir:

```
kume = set([14, 22, 'alfa', 'beta', 45])
```

Kümeler de listeler, demetler ve sözlükler gibi içlerinde farklı veri tiplerini barındırabilirler.

KÜMELERİN METOTLARI

İleri seviye veri yapılarında öğrendiğimiz diğer veri tipleri gibi kümelerin de çok önemli metotları mevcut. Bu metotlar yardımıyla kümelere ekleme, çıkarma ve diğer bir başka küme ile kıyaslamalar yapabiliyoruz.

İlk olarak hangi metotlarımız olduğunu öğrenelim.

Örnek kod

```
kume = set()
for i in dir(kume):
   if '__' not in i:
       print(i)
```

Çıktı

```
add
clear
copy
difference
difference_update
discard
intersection
intersection_update
isdisjoint
issubset
```

23

İleri Düzey Fonksiyonlar

Bu Bölümde

Bu bölüme kadar fonksiyonlar hakkında epey şey öğrendik. Nasıl tanımlanırlar, nasıl kullanılırlar, hazır fonksiyonlar nelerdir, neden fonksiyon kullanırız gibi pek çok sorunun cevabını biliyoruz. Ancak fonksiyonlar hakkında bu bildiklerimiz yeterli değil. Bazen işimizi kolaylaştırabilecek küçük numaralar yapmamız gerek. Oldukça az kod ile çok iş yapabilmemizin önünü açan bazı algoritma tekniklerini öğrenmemiz gerek.

Bu bölümde işte bu konulara değineceğiz.

LAMBDA FONKSİYONLARI

Eğer ki yazacağımız fonksiyon çok fazla kod içermiyor, yalnızca verilen parametrelere göre bir matematiksel sonucu döndürüryorsa normal fonksiyon tanımlamak yerine lambda fonksiyonu tanımlamak daha avantajlıdır. Hem tek satırda tanımlama yapılır hem de normal fonksiyonlara göre daha hızlı çalışır. Şimdi bu fonksiyonun nasıl tanımlandığını öğrenelim.

Normalde fonksiyon tanımlarken `def` anahtar sözcüğünü kullanırdık. Şimdi ise aşağıdaki gibi bir tanımlama yapacağız.

```
kare = lambda x:x*x
```

Tıpkı değişken tanımlar gibi fonksiyonun adını yazıp atama operatörü olan eşittiri yazacağız. Ardından lambda anahtar sözcüğünü yazıp fosnkiyonumuzun parametrelerini yazacağız. Yukarıdaki örnekte yalnızca bir parametre yazdık. Bu parametre `x` oluyor. **İki nokta** ifadesinden sonra ise bu fonksiyonun `return` edeceği değer giriliyor. Yukarıda bu değer `x*x` şeklinde ifade edildi. Yani verilen `x` değerinin karesini döndürüyor.

Daha başka örnekler yaparak lambda fonksiyonlarını pekiştirelim. Hatırlarsanız `filter` ve `map` fonksiyonlarını kullanırken yardımcı bir fonksiyondan faydalanıyorduk. Bu yardımcı fonksiyonu önceden tanımlamamız gerekiyordu. `Lambda` fonksiyonlarını kullanarak `map()` ya da `filter()` fonksiyonu ile ne yapacaksak yalnızca bir satırda yapabiliriz. Örneğin bir dizideki çift sayıları veren bir işlem yapmak için `filter()` fonksiyonunu kullanacağız. O halde dizideki eleman çiftse `True` değer döndüren bir fonksiyona ihtiyacımız var. Bunu `lambda` fonksiyonu ile yazmamız gerekirse şöyle yapabiliriz:

```
cift_mi = lambda x: x % 2 == 0
```

Eğer filter fonksiyonu ile bir satırda yazmamız ve kullanmamız gerekirse:

```
dizi = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
print(list(filter(lambda x: x % 2 == 0, dizi)))
```

İlk önce bir dizi tanımladık. İkinci satırda `print()` fonksiyonu içerisinde `list()` fonksiyonunu kullandık çünkü `filter()` fonksiyonundan dönen değer bir filter objesi. Bu objeyi bizim görebilmemiz açısından `list()` fonksiyonu ile listeye dönüştürdük. `filter()` fonksiyonunun ilk parametresine bir fonksiyon vermemiz gerekiyordu ve biz bunu `lambda` ile orada tanımladık. İkinci parametresine de ilk satırda tanımladığımız diziyi verdik.

Sonuca birlikte bakalım:

Çıktı

```
[2, 4, 6, 8, 10, 12, 14, 16]
```

Eğer lambda kullanmasaydık aynı kodu şu şekilde yazabilirdik:

```
dizi = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]

def cift_mi(x):
   return x % 2 == 0

print(list(filter(cift_mi, dizi)))
```

Her iki yöntem de doğrudur. Ancak `lambda` fonksiyonlarını kullanmanızı öneririm.

Bir de `map` fonksiyonu ile birlikte kullanalım lambda fonksiyonlarını. Bu defa dizideki tüm elemanların karelerinden oluşan bir dizi yaratalım.

Örnek kod

```
dizi = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
print(list(map(lambda x: pow(x,2), dizi)))
```

Çıktı

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256]
```

`map()` fonksiyonunun ilk parametresi olan yardımcı fonksiyonu da tek satırda tanımladık ve kullandık.

`Lambda` fonksiyonlarını yeri geldikçe kullanacağız. Ama şimdi fonksiyonlar ile kurabileceğimiz bir döngüyü öğreneceğiz. Bu döngü sayesinde çok fazla satır kod yerine çok ama çok az satır kod ile işimizi yapacağız.

Recursive (Özyinelemeli) Fonksiyonlar

Bu zamana kadar döngülerle pek çok iş yaptık. Döngüler için kısaca şunu söyleyebiliriz değil mi: bazı kodların belli sayıda ya da belli bir koşulda tekrar etmesi. Evet, döngüler için bu tanım şu an için yeterli. Yeni bir döngü öğrenmeyeceğiz ama döngü gibi bazı kodların belli bir şarta göre kendini tekrar etmesine yönelik algoritma biçimi öğreneceğiz.

26

Nesneye Tabanlı Programlamaya Giriş

Bu Bölümde

Bu bölümde sınıf ve obje (nesne) kavramlarının üzerinde durup bu ikisi arasındaki ilişkiyi inceleyeceğiz.

Sınıf kavramını bu zamana kadar incelememiş olsak da çok kez kullandık. Yapmadığımız tek şey yeni bir sınıf tanımlamak ve geliştirmek. Tıpkı herhangi bir amaç için fonksiyon tanımlayıp geliştirdiğimiz gibi bu defa sınıf tanımlayıp geliştireceğiz. İyi ama sınıf ne?

SINIF NEDİR?

Sınıf kavramını tek başına anlamak mümkün değil. Sınıfın paralelinde nesneyi de anlamak gerek. Eğer kabaca söyleyecek olursak sınıflar, nesne üretebilmek için nesnenin ne tür özelliklerinin olacağını, nesnenin nasıl çalışacağını Python'a anlattığımız bir yapıdır. Şimdilik bir taslak gibi düşünebilirsiniz. Yani üretimden çıkacak olan nesnenin taslağı gibi. Fakat bu düşüncenin yanlış olduğunu da bilin. Yalnızca şimdilik böyle düşünmenizde fayda var.

Bir sınıf tanımlayarak üretmek istediğimiz objeyi tanımlarız. Ve biz bu yöntemi çok kez kullandık aslında. Örneğin bir liste üretirken `list()` fonksiyonundan faydalanıyorduk ve bu fonksiyon ile aslında liste sınıfına ait bir örnek, bir obje üretiyorduk.

Sınıflar ile ilgili öğreneceğimiz pek çok konu var. Şimdiden tanımlama olayına girmezsek zaman kaybederiz. O halde bir sınıf nasıl tanımlanır öğrenelim.

```
class Yeni_Sınıf:
   pass
```

Bir fonksiyon tanımlanmasında `def` anahtar sözcüğünü kullandığımız gibi bir sınıf tanımlanmasında da `class` anahtar sözcüğünü kullanıyoruz. Sınıfa ait tüm bilgileri girinti içerisinde kodluyoruz. Buraya kadar anladık, tamam. İyi de sınıfın nasıl özellikleri olacak, bu özellikler nasıl kodlanacak? Daha da önemlisi biz bu sınıfı nasıl hayal edeceğiz de koda aktaracağız. Yani biz insanlar kodlamak için herhangi bir şeyi obje olarak nasıl görebileceğiz?

Yukarıdaki sorular bu konunun en önemli soruları. Bir programcı olarak yukarıdaki soruları yanıtlayamaz isek OOP felsefesini de kavrayamamış oluruz.

Herhangi bir objenin özelliklerini düşünelim. Örneğin bir kediyi ele alalım. Kedi bir canlıdır ama programcının kodları açısından yalnızca bir objedir. Bir bisikletten, gözlükten, arabadan ya da herhangi bir diğer şeyden farksızdır. Kedi, bir objedir. Kedinin özellikleri de objenin özellikleridir. Yani sınıf içerisinde tanımlanacak olan özelliklerdir.

Madem kedi üzerinden anlatıyoruz o halde sınıfın adını da kedi olarak değiştirelim.

```
class Kedi:
   pass
```

Umarım fark etmişsinizdir. Sınıf ismini büyük harfle başlatıyoruz. Evet, bu bir kültürdür. Ancak mecburiyet değildir. Küçük harflerle de yazsak Python açısından bir fark yoktur. Fakat Sınıf isimlerinin büyük harfle başlaması programcılık kültüründe yer

edinmiştir ve bunu bozmak bizim açımızdan da pek akıllıca davranış olmaz. Konuyu dağıtmamak adına bunun nedenini anlatmayı sonraya bırakıyoruz. Kedinin en belirgin özelliğinden yola çıkalım, miyavlamak. Kediler miyavlar. Bizim sınıfımızda da kedi miyavlamalı. Fakat buna gelmeden kedinin fiziksel birtakım özelliklerini belirtelim. Örneğin kedimiz hangi renk, kaç kilogram gibi soruların yanıtlarını verelim.

```
class Kedi:
   renk = "Siyah"
   kg = 1.5
```

Tüm tanımımızın bu kadar olduğunu varsayarsak bu bilgilere erişmeyi nasıl başaracağız? Yani bu sınıfın renk değerine nasıl ulaşabiliriz?

Örnek kod

```
class Kedi:
   renk = "Siyah"
   kg = 1.5

print(Kedi.renk)
```

Çıktı

```
Siyah
```

Aynı mantık ile kedinin kütlesine ulaşalım.

Örnek kod

```
print(Kedi.kg)
```

Çıktı

```
1.5
```

Listelerde, demetlerde, sözlüklerde ve kümelerde nokta koyduktan sonra metot çağrıyorduk. Aslında bu tüm objeler için geçerli. İster metodu ister objeye ait bir değişkeni çağrılabiliriz. Yukarıda değişken çağırmayı gördük. Ancak henüz ortada bir obje yok. Yalnızca bir sınıf var.

Eğer ki üretilmesi gereken obje yalnızca bir tane ise örnekleme yapmadan yalnızca sınıf üzerinden işlem yapılabilir. Mesela bu kediye ait bilgilerde oynama yapalım ve herhangi bir örnekleme yapmadan bunu yalnız sınıf üzerinden gerçekleştirelim.

29

Fonksiyonlarda İleri Seviye Konular

Bu Bölümde

Bu bölümde fonksiyonlar hakkında bilmediğimiz konuları -ki bu konular şunlardır: iç içe fonksiyonlar, obje olarak fonksiyon, *args ve **kwargs parametreleri, decorator (bezeyici)- öğreneceğiz.

İÇ İÇE FONKSİYONLAR

Fonksiyonları nasıl tanımlayacağımızı, isimli ya da isimsiz parametreleri nasıl vereceğimizi ve nihayetinde nasıl kullanacağımızı biliyoruz. Ancak fonksiyonlar bundan çok daha fazlasını yapabilirler. Bu konuyu NTP içerisine dahil ediyoruz çünkü bu defa fonksiyonu bir obje olarak öğreneceğiz. Fonksiyonların birer nesne olmasından dolayı ne gibi özellikleri olabileceğini göreceğiz. Python'da fonksiyonlar da birer obje ise objelerin temelde sahip oldukları tüm özelliklere fonksiyonlarda sahip demektir. O halde bir objenin temelde ne gibi özellikleri olabileceğinden bahsedelim.

Bir objeyi değişkende tutabiliyoruz değil mi? Örneğin karakter dizisini bir değişkene atayabiliriz. Fonksiyonlar için de geçerli bir durum o halde. Bir fonksiyonu herhangi bir değişkene atayabilir ve bu değişkeni de dilediğimiz fonksiyona parametre olarak verebiliriz. Bir örnek yapalım.

```
def func(a,b):
   return sum((a,b))

x = func
```

Bundan sonra `func()` fonksiyonunu `x()` yazarak da çağırabiliriz. Dikkat ederseniz fonksiyonu çağırmadık. Yalnızca bir değişkene atadık. Çağrılma durumunda tıpkı bir objenin örneklenmesi gibi sonuna parantezlerin konulması gerekir.

Tanımladığımız fonksiyonu `x()` şeklinde çağıralım ve sonucu görelim.

Örnek kod

```
def func(a,b):
   return sum((a,b))

x = func
print(x(2,3))
```

Çıktı

```
5
```

Listeleri hatırlıyorsunuz değil mi? Bir listeyi bir başka değişkene atadığımızda bellekte yeni bir adresleme yapmıyorduk.

Örneğin;

Örnek kod

```
liste1 = [1,2,3,4,5]
liste2 = liste1

print("Liste 1 ID:", id(liste1))
print("Liste 2 ID:", id(liste2))
```

Çıktı

```
Liste 1 ID: 140470263553928
Liste 2 ID: 140470263553928
```

Yukarıdaki durum fonksiyonları bir değişkene atarken de geçerli olacak mı görelim.

Örnek kod

```
def func(a,b):
   return sum((a,b))

x = func

print('X ID:',id(x))
print('func ID:',id(func))
```

Çıktı

```
X ID: 140176700137952
func ID: 140176700137952
```

Anlaşılıyor ki herhangi bir yeni adresleme yapmıyoruz. 'x' adında bir değişkene atama yaparken x değişkenini yalnızca bellekte var olan bir adresi gösteren yeni bir değişken haline getiriyoruz. Fonksiyonlar birer nesne olduğuna göre return ile de kullanılabilirler. Yani bir fonksiyonun geriye döndürdüğü değer yine bir fonksiyon olabilir. Bunun üzerinde de birkaç örnek yapacağız. Ancak asıl gelmeye çalıştığımız esas nokta decorators (bezeyiciler) konusudur. Fakat önce birkaç örnek yapalım ve bir fonksiyonu obje olarak kullanmaya alışalım.

Geriye bir fonksiyon döndüren bir fonksiyon yazalım. Kurgumuz şu: yazacak olduğumuz fonksiyon bizden bir parametre alacak. Bu parametre 'toplama', 'çıkarma', 'çarpma' veya 'bölme' olabilir. Yazdığımız parametreye göre geriye bu dört işlemden birini

33

Glue

Bu Bölümde

Bu bölümde Glue adında, önceki iki projenin devamı niteliğinde ve önceki iki projenin kullanılmasından sonra kullanılabilecek olan bölünmüş dosyaları birleştiren bir program yazacağız.

GLUE NEDİR?

Glue projesi, bölünmüş dosyaları (tek bir dosyadan bölünmeseler bile) birleştirmek için yazacağımız bir program olacak. Proje başlığı altında neler yaptığımızı hatırlayalım: Dosyaları tarayıp kaç satır, kelime ve karakterden oluştuğu bilgisini veren Word Counter projesini yaptık.

Dosyaları satır, kelime ve karakter sayılarına göre bölen Splitter projesini yaptık.

Şimdi de bölünmüş dosyaları tekrar bir bütün haline getiren Glue projesini yapıyoruz.

> **NOT** Projenin tüm kodlarına şu adresten ulaşabilirsiniz:
> ***https://github.com/volkantasci/PythonEgitimKitabi/blob/master/glue.py***

PROGRAMIN YAZILMASI

Glue, Türkçe manasıyla yapıştırıcı programı belirlenmemiş sayıda argüman alabilmeli. Nihayetinde kullanıcının kaç dosyayı birleştirmek isteyeceğini bilmiyoruz. Dolayısı ile argüman sayısına sınırlama koyamayız. Ancak argümanların sırasını belirleyebiliriz.

İlk sırada zaten dosyanın kendisi yer alıyor. Son sıraya da sonuçta ortaya çıkacak olan bütün haldeki dosyanın adını yazalım. Örneğin kullanımı şöyle olsun:

```
glue file1 file2 file3 result
```

`file1`, `file2` ve `file3` dosyalarını birleştirip result adında yeni bir dosyaya yazacak. Sonuç dosyasını en sona yazmamız doğrultusunda `argv[-1]` şeklinde bir kullanımla gerekli dosya adına ulaşmış oluruz. Programımızı oluşturmaya başlayalım.

```
#!/usr/bin/python

from os.path import isfile, exists
from sys import argv
```

Artık bu satırları yazmaya alıştık sanırım. Program içerisinde kullanacağımız fonksiyonları şimdiden import ettik. Metotlarımızın `@staticmethod` bezeyicisi ile bezeneceğini söylememize gerek yoktur sanırım.

```
class Glue:
   @staticmethod
   def help_page():
       msg = """
```

```
        Glue, birden çok dosyayı birleştirip tek dosya haline getirmek için
kullanılan programdır.
        Python ile yazılmıştır.

        [EXAMPLE]:
        --glue file1 file2 file3 result
        """

        return msg
```

Tüm projelerimizde kullandığımız gibi yine bir yardım sayfası oluşturduk. Biraz önce kullanımın nasıl olması gerektiğinden söz etmiştik. Yardım sayfasında da bu kullanımı gösterdik. Kullanıcı kaç adet dosya girerse girsin hepsinin kontrolünü yapmalı ve bir sorun gördüğümüzde bunu kullanıcıya bildirmeliyiz. Sorunlu bir dosya ile birleştirme işlemi yapmamalıyız.

```
    @staticmethod
    def check_files(files):
        for i in files:
            if exists(i):
                return isfile(i)
            else:
                return False

        return True
```

Dosyaları sırayla kontrol eden bir metot yazdık. Eğer aralarından birisi mevcut değil ya da mevcut ama dosya değilse geriye False değeri dönen bir metot. `main()` fonksiyonu içerisinde kullanarak dosyaların doğruluğunu teyit etmemize yarayacak.

```
    @staticmethod
    def write_result_file(files, result):
        x = ""
        for i in files:
            with open(i) as f:
                x += f.read()

        with open(result, 'w') as rf:
            rf.write(x)
```