

Veri Bilimi & Python

Hafta 2

Doç. Dr. Deniz KILINÇ
Murat ŞAHİN

Veri Nedir?

Veri işlenmemiş ham bilgidir. Tek başına bir anlam ifade etmez.

Veri üreten kaynaklar her yerdedir; sosyal medya kanalları, IoT, ülkelerin savunma sanayisinde kullandıkları araçlar, insan.

Veri 21. yüzyılın yeni petrolü (ham) olarak nitelendirilmektedir [1].

Veriye ile temasta olmayan hiçbir ihtiyatlı meslek yoktur, varsa da yok olacaktır.



Veri Nedir? (devam...)

İnsan var olduđu andan itibaren veri üretmektedir. Bu bilinçsizce üretim, milattan önceki dönemlere ait kalıntılardan başlayıp, bugün parmaklarımızın ucundaki “like”larımıza (sosyal medyadaki beğenilerimiz ve paylaşımlarımız) kadar uzanmakta, çoğalmakta çeşitlenmektedir.

Bu küçük parçanın/zerrenin (verinin) uzun yolculuğuna karşın, “veri (data)” olarak adlandırılması ve gücünün anlaşılması sanıldığı uzun zaman almıştır.



DIKW Primadi

İngilizce Data (veri), Information (enformasyon), Knowledge (bilgi), Wisdom (bilgelik) kavramlarının baş harfleri olan DIKW terimi bir veri ilerleme/dönüşüm piramidi olarak literatürde yerini almıştır.

Şekil’de de görüldüğü üzere piramidin en altından en üstüne doğru, elimizdeki parçacıklar/veriler çeşitli analiz işlemlerinden sonra miktar olarak azalmakta ve bir üst aşamaya geçmektedir.

Öte yandan aşağıdan yukarıya her geçişte (dönüşümde) ise elde edilen “değer” (anlamsallık) artmaktadır.



Veri Bilimi Nedir?

Karmaşık problemleri çözmek için hem yapılandırılmış hem de yapılandırılmamış veriyi, işe yarar/değerli bilgiye (knowledge) dönüştüren disiplindir.

Bilimsel problem tanımlama ve çözme yöntemlerinin, matematiğin, istatistiğin, yazılım geliştirmenin ve teknolojinin oluşan **çok disiplinli** bir bilim dalıdır.

Veri bilimi, şirketlerin gelir ve performans arttırma amaçlarının anlaşılması ve bu doğrultuda problemleri için bazı araç ve algoritmaların yaklaşımları geliştirilmesi sürecidir.

Data Scientist: *The Sexiest Job of the 21st Century*

*Meet the people who
can coax treasure out of
messy, unstructured data.*
by Thomas H. Davenport
and D.J. Patil

When Jonathan Goldman arrived for work in June 2006 at LinkedIn, the business networking site, the place still felt like a start-up. The company had just under 8 million accounts, and the number was growing quickly as existing members invited their friends and colleagues to join. But users weren't seeking out connections with the people who were already on the site at the rate executives had expected. Something was apparently missing in the social experience. As one LinkedIn manager put it, "It was like arriving at a conference reception and realizing you don't know anyone. So you just stand in the corner sipping your drink—and you probably leave early."

Veri Bilimci Kimdir?

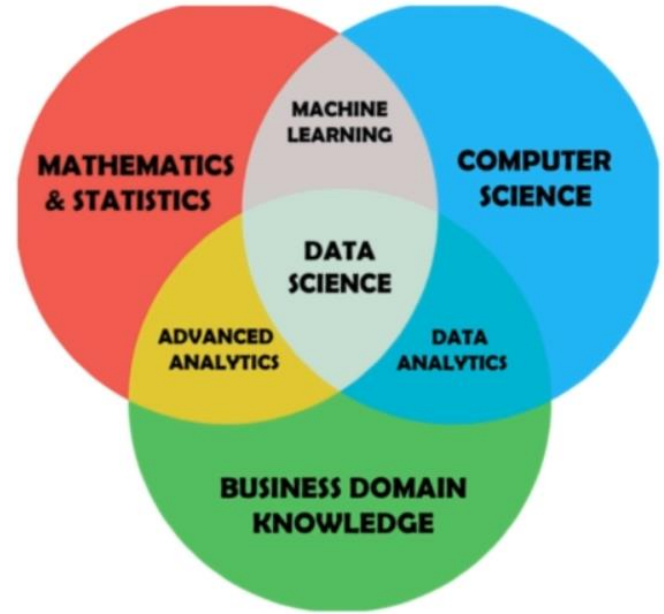
Her türlü araç ve bilimsel tekniğin kullanılarak veriden faydalı bilgiler, aksiyon tavsiyeleri, karar destek sistemleri ve veri odaklı ürünler ortaya çıkaran kişidir.



irfan bulu

Deep Learning Scientist | Senior Physicist @ Schlumberger
4d • Edited

"I am a data scientist. What that means is: the worst programmer in a room full of programmers, the worst statistician in a room full of statisticians and the worst basic science person in a room full of scientists" . I really like this quote from Christopher Fonnesbeck 😊 And that's absolutely a great thing! That means we incorporate a lot of ideas from very different fields in to our work and thinking!



4 Aşamada Veri Bilimi Projesi

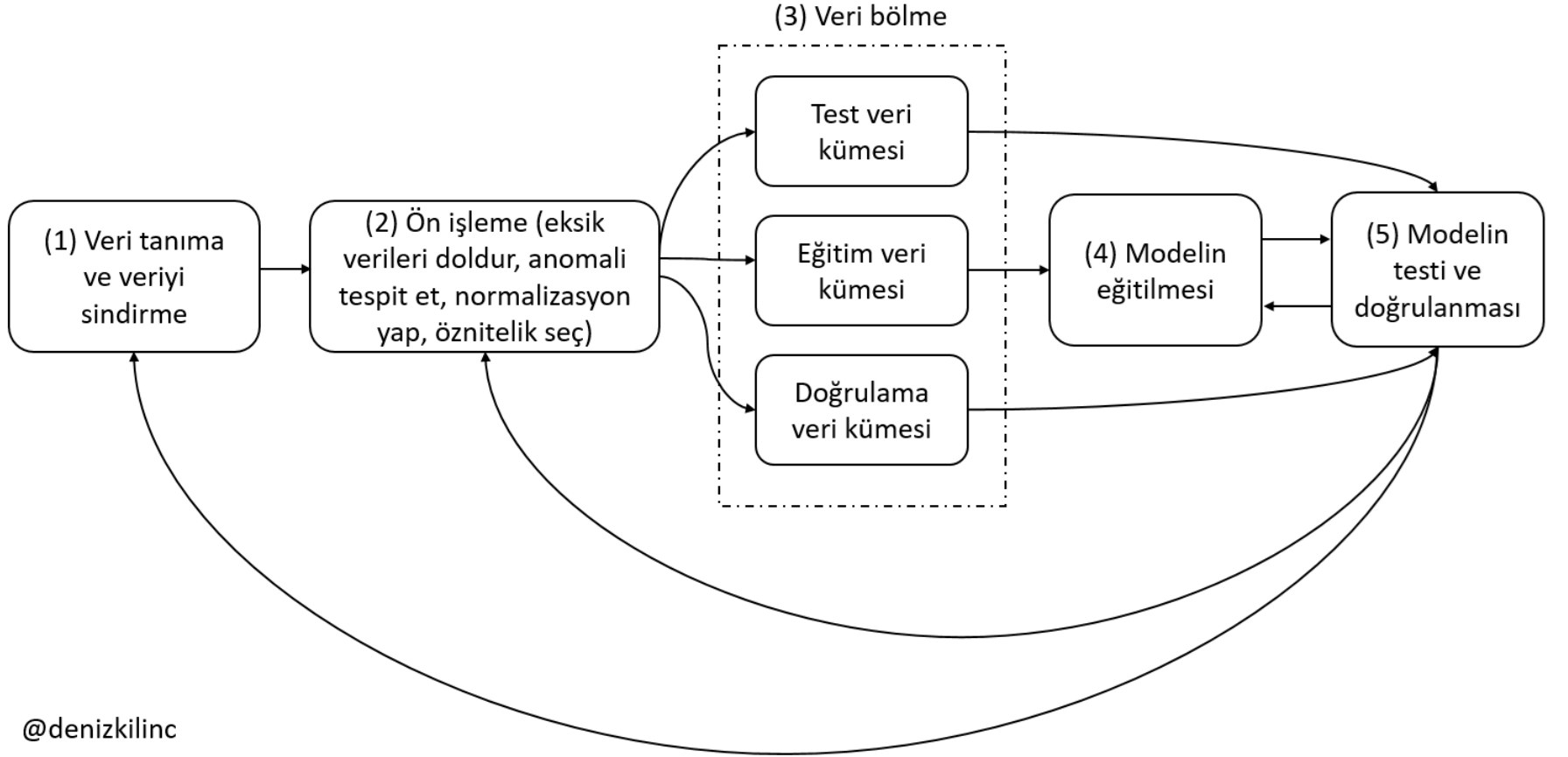
Veri bilimi problemleri için farklı yaklaşımlar olsa da aşağıdaki gibi 4 temel aşamaya sahip bir yol haritası olduğunu söylemek mümkündür.

1. Problem Tanımlama
2. Veri Hazırlama
3. Model/Algoritma Seçimi
4. Sonuçları İyileştirme

Bilinen Veri Bilimi Metodolojileri

Data Mining Process Models	KDD	CRISP-DM	SEMMA
No. of Steps	9	6	5
Name of Steps	Developing and Understanding of the Application	Business Understanding	-----
	Creating a Target Data Set	Data Understanding	Sample
	Data Cleaning and Pre-processing		Explore
	Data Transformation	Data Preparation	Modify
	Choosing the suitable Data Mining Task	Modeling	Model
	Choosing the suitable Data Mining Algorithm		
	Employing Data Mining Algorithm		
	Interpreting Mined Patterns	Evaluation	Assessment
	Using Discovered Knowledge	Deployment	-----

(Cross Industry Standard Process Model for Data Mining)



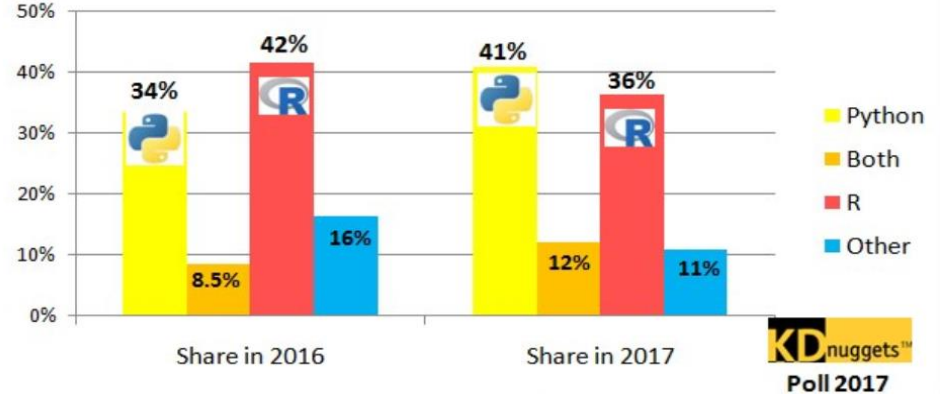
Python Programlama Dili

Günümüzde veri bilimi denilince akla ilk gelen programlama dili Python'dır.

Python, Guido van Rossum tarafından 1990 yılında tasarlanmış bir programlama dilidir.

İsmi Rossum'un çok sevdiği Monty Python komedi grubunun oynadığı, **Monty Python's Flying Circus** isimli gösteriden gelmektedir.

Python, R, Both, or Other platforms for Analytics, Data Science, Machine Learning



Python Kodlamak

Açık kaynak kod lisansına sahip olan ve ücretsiz yazılım geliştirilmesine imkan veren Python programlama dilinin; Windows, Unix/Linux ve MacOS işletim sistemleri üzerinde farklı yöntemlerle çalıştırılması mümkündür.

Programlama dil yapısı açısından incelendiğinde, yüksek seviyeli diller kategorisine giren Python'un en ilginç özelliği;

- i) fonksiyonel programlama,
- ii) nesne yönelimli programlama ve
- iii) yapısal programlama dil paradigmalarının

hepsini desteklemesidir. Dilin diğer güçlü yanlarından birisi de dinamik dil yorumlayıcısına (interpreter) sahip olmasıdır.

Python Kodlamak

Python dili ile yazılmış bir programın bilgisayarınızda çalışabilmesi için işletim sistemiyle uyumlu Python yorumlayıcının bilgisayarınızda kurulu olması gerekmektedir. Resmi Python sitesinden uygun yorumlayıcıya sahip kurulum paketini indirip, sisteminize kurabilirsiniz.

Python kodlamak için Notepad++, Spyder, VS Code, PyCharm gibi pek çok alternatif mevcuttur.

Daha efektif ve anlaşılır çalışmalarda bulunmak için derslerde Anaconda Navigator bünyesinde bulunan Jupyter Notebook kullanacağız.



Merhaba Dünya!

Bir mesaj yazdırmak için "print()" fonksiyonu kullanılabilir. Parametre olarak yazdırılacak değeri alır.

```
print("Data never sleeps!")
```

```
Data never sleeps!
```

Girinti Hassasiyeti

Python diğer dillerin aksine girintilere duyarlıdır.

```
print("Data never sleeps.")  
print("Data is the new oil.")
```

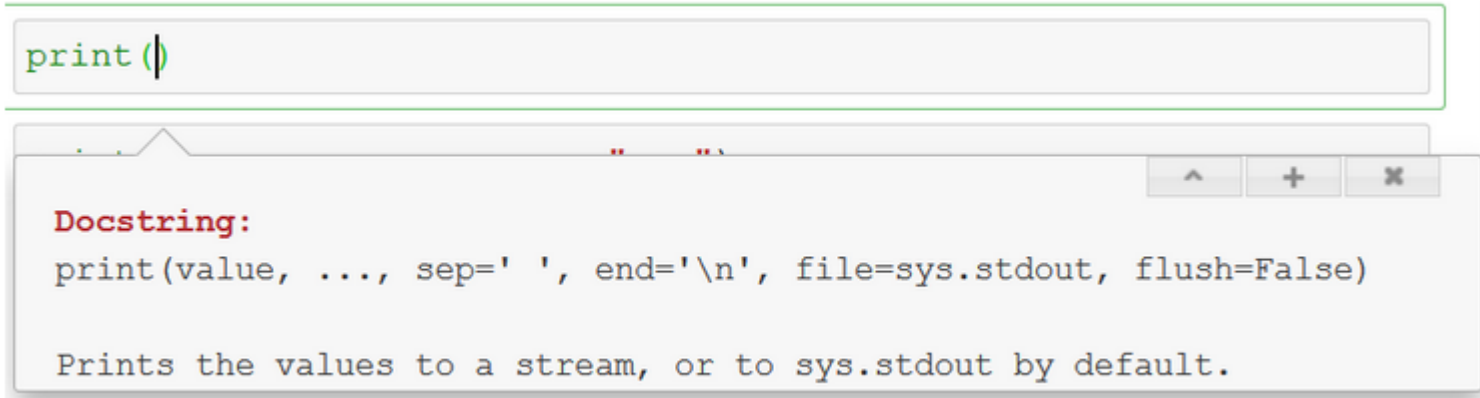
File "<ipython-input-30-e53a611aee9d>", line 2

```
    print("Data is the new oil.")  
    ^
```

IndentationError: unexpected indent

Çevrimdışı Yardım Almak

Herhangi bir fonksiyon yazdığınızda eğer aklınıza hangi parametreler aldığı gelmiyorsa ve üstelik internetiniz yoksa yine de yardım alabilirsiniz. Parantezin içinde "Shift" ve "Tab" tuşlarına basılı tutarak yardım alabilirsiniz.



The screenshot shows a code editor with a light gray background. At the top, a text input field contains the code `print ()` with a green cursor at the end. Below this, a dropdown menu is open, displaying the docstring for the `print` function. The docstring is formatted with a red header **Docstring:**, followed by the function signature `print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)` in a monospaced font. Below the signature, a descriptive sentence reads: "Prints the values to a stream, or to sys.stdout by default." The dropdown menu has a standard window control bar with minimize, maximize, and close buttons.

```
print ()
```

Docstring:
`print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)`
Prints the values to a stream, or to sys.stdout by default.

Python Değişken Tipleri

Python'da değişken tanımı, değişkene değer atandığında otomatik olarak gerçekleşir. (*type inferred language*)

Sağ şekilde görüntülenen değişken tipleri Python'da sıklıkla kullanılan değişken tiplerdir.

Text Type:

`str`

Numeric Types:

`int`, `float`, `complex`

Sequence Types:

`list`, `tuple`, `range`

Mapping Type:

`dict`

Set Types:

`set`, `frozenset`

Boolean Type:

`bool`

Stringler

Karakter dizilerini tutmak için oluşturulan değişken tipidir. Python'da stringler tırnak içinde temsil edilir. Stringlerin dilimlenmesi de listelere oldukça benzer.

Bir string değişkenin kaç karakterden oluştuğunu `length`'i temsil eden bir fonksiyon ile görebiliriz.

```
: cumle = "Veri Bilimi Eğitimi."
```

```
: len(cumle)  
# tüm boşlukları ve noktalama işaretlerini de kapsar.
```

```
: 20
```

Bazı String Fonksiyonları

Sadece stringler için yüzlerce hazır fonksiyon vardır. Python metin işleme ve doğal dil işleme konuları için idealdir.

```
cumle.upper()
```

```
'VERI BILIMI EĞİTİMİ.'
```

```
cumle.lower()
```

```
'veri bilimi eğitimi.'
```

```
cumle.capitalize()
```

```
'Veri bilimi eğitimi.'
```

Boolean Değişken Tipi

“Doğru” ve “Yanlış” olmak üzere iki değer alır. Kontrol işlemlerinde kullanılır.

```
degisken1 = True  
degisken2 = False
```

```
degisken1 == degisken2 # eşit değildir birbirlerine.
```

False

```
degisken1 != degisken2 # eşit değiller mi?
```

True

```
print(10 > 5) # 10 büyük müdür 5 den ? Evet.
```

True

Liste Değişken Tipi

Birden fazla değeri tutmak için tasarlanmıştır. Listenin her elemanı farklı bir veri tipi içerebilir, bu yüzden oldukça esnektir. Esnekliğin bedeli vardır, maliyetlidir.

```
liste2 = ["elma", 25, True, 0, 3.1689, False]  
  
print(liste2)
```

```
['elma', 25, True, 0, 3.1689, False]
```

Listelerde Dilimleme İşlemleri

Listenin elemanlarına erişmek için indis numaralarını vermek gerekir. Liste indisleri sıfırdan başlar. Tek bir elemana erişmek için o elemanın indisini vermek yeterlidir.

Eğer listenin ilk üç elemanı veya ortadan iki elemanı gibi daha spesifik aralıkları seçmek istersek ":" operatörünü kullanmalıyız.

L[start:stop:step]

Start position

End position

The increment

Listelerde Dilimleme İşlemleri

```
liste2[2 : 5]  
# listenin 2. 3. ve 4. elemanlarını gösterir.
```

```
[True, 0, 3.1689]
```

```
liste2[1 : 3]  
# listenin 1. elemanından başlar ve 3. elemana kadar gösterir.
```

```
[25, True]
```

```
liste2[: 3]  
# listenin başından başlar, 3. elemana kadar gösterir.
```

```
['elma', 25, True]
```

Bazı Liste Fonksiyonları

```
liste2.append("yeni")  
print(liste2)
```

```
['elma', 25, True, 0, 3.1689, False, 'yeni']
```

Peki ya birinci indisten sonrasına yeni nesne eklemek istersek?

```
liste2.insert(1, "yeni2")  
print(liste2)
```

```
['elma', 'yeni2', 25, True, 0, 3.1689, False, 'yeni']
```

Yeni eklediğimiz nesneyi silelim.

```
liste2.remove("yeni2")  
print(liste2)
```

```
['elma', 25, True, 0, 3.1689, False, 'yeni']
```

Doğru Kopyalama İşlemi

```
liste = [10,56,8,91,25,78,63]
kopya = liste.copy()

print(liste)
print(kopya)
```

```
[10, 56, 8, 91, 25, 78, 63]
[10, 56, 8, 91, 25, 78, 63]
```

```
del kopya[0]

print(kopya)
print(liste)
```

```
[56, 8, 91, 25, 78, 63]
[10, 56, 8, 91, 25, 78, 63]
```


Referans ve Adres

Python programlama dilindeki birçok nesne türü (örneğin: listeler) *referans* türündedir (reference type).

Bu durumda, bir nesneye atama yaparken “ilgili değerleri” değil, “değerlerin tutulduğu bellek adresini” kullanırsınız.

İki listenin aynı adresi gösterip göstermediğini nereden anlarız?

```
kopya is liste    # False dönerse aynı adresi göstermiyor,  
# True dönerse aynı adresi gösteriyor.
```

False

Hatalı Kopyalama İşlemi

Dolayısıyla referans türündeki bir nesneyi, başka bir nesneye “=” operatörü ile atayarak, yeni değerlere sahip bir nesne oluşturmuş (kopyalamış / klonlamış) olmazsınız, aynı bellek adresini gösteren “ikinci nesneniz” olur.

Sonuç itibari ile bir nesnede yapılan herhangi bir değişiklik diğer nesnede de aynı şekilde etkisini gösterir.

```
kopya = liste
```

```
del kopya[0]  # kopya listenin ilk elemanını uçuralım.
```

```
print(kopya)  # ve yazdıralım.
```

```
print(liste)  # listeyi de yazdıralım.
```

```
[56, 8, 91, 25, 78, 63]
```

```
[56, 8, 91, 25, 78, 63]
```

Sözlük Değişken Tipi

Sözlükler anahtar-değer çiftlerini saklayabilen özel koleksiyonlar olup, anahtarların mutlaka benzersiz değerlere sahip olması gerekmektedir.

Sözlük oluşturulması sırasında süslü parantezler kullanılmaktadır.

```
sozluk = {  
    "marka": "Ford",  
    "model": "Mustang",  
    "uretim_yili": 1964  
}  
  
print(sozluk)
```

```
{'marka': 'Ford', 'model': 'Mustang', 'uretim_yili': 1964}
```

Sözlük Değişken Tipi

Sözlük yapısı, veri çerçevelerine benzediği için önem teşkil eder.

```
cocuklarim = {  
    "cocuk1" : {  
        "ad" : "Ali",  
        "yil" : 2007  
    },  
    "cocuk2" : {  
        "ad" : "Veli",  
        "yil" : 2003  
    },  
    "cocuk3" : {  
        "ad" : "Hüseyin",  
        "yil" : 1999  
    }  
}
```

Bazı Sözlük Fonksiyonları

```
#sözlük kullanımı
notlar = {
    "0001-Ada": 83,
    "0002-Cem": 79,
    "0003-Sibel" : 82
}

#sözlük elemanına erişim
notlar["0001-Ada"] #83
```

83

```
notlar.keys() # sadece anahtarları gösterir.
```

```
dict_keys(['0002-Cem', '0003-Sibel', '0004-Nil'])
```

```
notlar.items() # anahtarları ve değerleri gösterir.
```

```
dict_items([('0002-Cem', 79), ('0003-Sibel', 82), ('0004-Nil', 99)])
```

```
notlar.values() # sadece değerleri gösterir.
```

```
dict_values([79, 82, 99])
```

Koşullu İfadeler

Koşul deyimi olarak “if” kullanılmaktadır. Koşul sonucu doğru (true) ise koşul devamındaki iç içe yuvalanmış kodlar işletilir, doğru değilse (false) “else” deyimi ve devamındaki kod satırları çalışır.

Ayrıca “elif” deyimi kullanılarak, else kod blokuna düşülen bir durumda, ilave bir “if” deyimi daha çalıştırılabilir.

Koşullu İfadeler

Koşullu ifadelerin kullanımına bir örnek kod bloğu verilmiştir.

```
b, c = 200, 250

if b > c:
    print("b büyüktür c'den.")

else: # if bloğuna girmezse kesinlikle buraya giriyor.
    print("b büyük değildir c'den.")
```

b büyük değildir c'den.

İç İçe Koşullu İfadeler

```
x = 41

# eğer ilk if koşulu sağlanmazsa direkt bitirecek.

if x > 10:
    print("x 10'dan büyüktür.")

    if x > 20:
        print("ve x 20'den de büyüktür!")

    else:
        print("ve x 20'den büyük değildir.")
```

x 10'dan büyüktür.
ve x 20'den de büyüktür!

Python ile Döngüler

Döngü; bir kod bloğunun istenildiği kadar tekrar edilmesini sağlar.

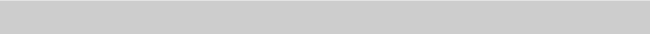
Python'da da *while* ve *for* olmak üzere iki temel döngü türü bulunmaktadır.

While döngüsü, belirttiğiniz koşul geçerli olduğu sürece bir kod bloğunun çalıştırılmasını sağlar. Sonsuz döngüye girmesi kolaydır.

For döngüsü ise döngünüzün kaç defa dönmesini gerektiğini biliyorsanız veya sıralı bir dizinin tüm elemanlarına işlem uygulamak istiyorsanız tercih edilebilir.

While Döngüsü

```
sayac = 0  # sayaca başlangıçta 1 değerini verdik.  
  
while sayac < 6:    # döngüye girmenin şartı sayacın 6 değerinden  
    print(sayac)    # sayacı her döngü aşamasında bastırıyoruz.  
    sayac += 1      # her seferinde sayacın değerini 1 artırıyoruz.  
  
# eğer sayacın değerini 1 artırmazsak asla 6 değerine ulaşamayacağız  
# sonsuz bir döngüye girmiş olur ve çekirdek kilitlenebilir.
```

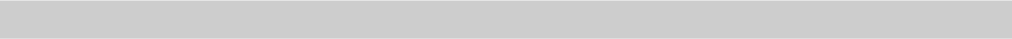
<  >

0
1
2
3
4
5

For Döngüsü

For döngüsü herhangi bir listenin elemanlarını bastırmak için oldukça idealdir

```
meyve_listesi = ["elma", "armut", "kiraz", "kavun"]  
  
for meyve in meyve_listesi: # meyve listesindeki her bir nesneye '  
    print(meyve)             # her bir nesneyi yani meyveyi de bastır
```

```
<  >  
elma  
armut  
kiraz  
kavun
```

For Döngüsü

Bu yapı listelere oldukça benzer. `range()` fonksiyonu başlangıç, bitiş ve opsiyonel olarak artırma değeri ile 3 parametre alır. Bir sayı dizisi oluşturur.

```
for x in range(7):  
    print(x)  
  
else:  
    print("Nihayet bitti!")
```

```
0  
1  
2  
3  
4  
5  
6  
Nihayet bitti!
```

Fonksiyonlar

Fonksiyonlar belirli bir isme sahip program parçalarıdır.

Karmaşık yapıdaki programların karmaşıklığını azaltmak ve bu programları modüler bir yapıya kavuşturmak için kullanılırlar.

Modüler yapıları sayesinde programcıların tekrarlanan kodlar yazmalarını önlerler.

Python'da fonksiyon tanımlayıcı anahtar kelime olarak “def” kullanılır.

Geriye bir değer döndürülmesi isteniyorsa “return <deger>” şeklinde bir kod satırı ile fonksiyon tamamlanır.

Fonksiyonlar | Python

Parametre almayan ve geriye değer döndürmeyen bir fonksiyon örneği.

```
def selamla(): # parametre almamış, bir veri döndürmeyen fonksiyon  
    print("Selamlar!") # mesaj yazdırıyor.
```

```
selamla()
```

Selamlar!

Fonksiyonlar | Python

Parametre alan ve geriye değer döndüren bir fonksiyon örneği.

```
def topla(x,y): # iki değeri parametre olarak alır  
    return x+y # parametreleri birbiriyle toplar ve değer döndürür.  
  
topla(5,10)
```

Fonksiyonlar | Python

Varsayılan parametre alan ve geriye değer döndüren bir fonksiyon örneği.

```
def topla(x = 10, y = 20): # Eğer değer girilmezse diye varsayılan parametre değerlerini girdik.  
    return x + y # iki değer toplamını döndürür.  
  
topla(50) # ilk parametre olan x değerini girdik sadece  
         # ikincisini girmedığımız için varsayılan 20 aldı.
```

70

```
topla(y = 44) # ikincisini girdik, bu sefer x değerini varsayılan olarak 10 aldı.
```

54

Hiçbir parametre girmesek ? Varsayılan parametrelerin toplamını döndürür.

Lambda Fonksiyonları | Python

Python programlama dilindeki anonim lambda fonksiyonları, istenildiği kadar girdi parametresi alır ve bir tane deyimi çalıştırır. Tek satırda yazılması ve performanslı olmasından ötürü tercih sebebidir.

```
x = lambda a : a * 10  
x(5)
```

50

```
y = lambda a, b : a + b  
y(50, 65)
```

115

Lambda Fonksiyonları | Python

fnc lambda fonksiyonu 1 adet parametre alarak, parametrenin değeri 1 artırıp geri dönmektedir.

fnc2 lambda fonksiyonu ise aldığı 2 adet parametrenin değerini toplayarak geri dönmektedir.

Dikkat edilirse bir lambda fonksiyonunun değeri olarak başka bir fonksiyon kullanmak mümkündür.

```
fnc = lambda x : x + 1  
print(fnc(1))  
print(fnc(fnc(1)))
```

2
3

```
fnc2 = lambda x,y : x + y  
  
print(fnc2(4,7))  
print(fnc2(4, fnc(1)))
```

11
6

Lambda Fonksiyonları | Python

Lambda fonksiyonları içeren fonksiyonlar yazarak generic(genel) fonksiyonlar oluşturmak da mümkündür.

```
# lambda fonksiyon içeren fonksiyon tanımları

def fnc3(n):
    return lambda x : x ** n

fnc_kare = fnc3(2) # dinamik kare alma fonksiyonu
fnc_kup = fnc3(3)  # dinamik küp alma fonksiyonu

print(fnc_kare(5)) # 25
print(fnc_kup(4))  # 64
```

25

64

Python ile Nesne Yönelimli Programlama

Nesne yönelimli programlama (Object Oriented Programming) nesneyi merkezine alan bir bilgisayar programlama yaklaşımıdır.

Nesneler/varlıklar ve onların sahip oldukları özellikler üzerinde durulur.

Fonksiyonların kullanıldığı yapısal programlama yaklaşımında sadece soruna odaklı farklı fonksiyonlar yazılır ve sadece o sorun için fonksiyonlar kullanılır.

Yapısal programlama yaklaşımıyla oluşturulmuş bir program binlerce ayrı isimde tanımlanmış değişken ve yüzlerce farklı fonksiyona sahip olabilir.

En ufak işlem için bile ayrı fonksiyon oluşturulması gerekir ve bu programın karmaşıklığını arttırır.

Python ile Nesne Yönelimli Programlama

Nesne yönelimli programlamanın temel kavramları;

- Büyük yazılımlar geliştirmeyi kolaylaştıran **Soyutlama (Abstraction)**,
- Yazılımları değiştirmeyi ve korumayı kolaylaştıran **Saklama/Sarmalama (Encapsulation)**,
- Yazılımları genişletilebilir kılan **Sınıf Hiyerarşisi ve Kalıtım (Inheritance)**,
- Kalıtım hiyerarşisi içerisinde farklı işlemler yapan (fakat) aynı isimdeki özellik veya metotların kullanımına **Çok Biçimlilik (Polymorphism)**

Sınıf | Python ile NYP

Nesnelerimizi oluşturmak için öncelikle class anahtar kelimesi ile yeni bir sınıf oluşturuyoruz. Burada araba sınıfı adı altında nesneler oluşturmak için öncelikle araba sınıfını oluşturmalıyız.

Araba için hangi özellikleri uygun görüyorsak bunlarla bir sınıf oluşturabiliriz.

```
class araba():  
  
    def __init__(self, marka, renk, silindir):  
        self.marka = marka  
        self.renk = renk  
        self.silindir = silindir
```

Nesne | Python ile NYP

Sınıfların tanımlanmasında “class” anahtar kelimesi ve kurucu metot (constructor) tanımında “__init__” fonksiyon adı kullanılmıştır.

self anahtar kelimesi kullanılarak o anki aktif nesnenin özelliklerine ve metotlarına erişilebilir.

Üst sınıfa (parent) erişmek için ise `super()` anahtar kelimesi kullanılabilir.

```
class insan():  
  
    def __init__(self, boy, kilo, goz_rengi):  
  
        print("insan nesnesi yaratılıyor.")  
  
        self.boy = boy  
        self.kilo = kilo  
        self.goz_rengi = goz_rengi
```

```
murat = insan(boy = 182, kilo = 83.5, goz_rengi = "yesil")  
  
insan nesnesi yaratılıyor.
```

```
print(murat.boy)
```

182

Sınıf ve Metodu | Python ile NYP

Bu sefer oluşturduğumuz sınıfa bir **fonksiyon** ekledik.

Sınıftan “evim” nesnesini belirlediğimiz özelliklerle oluşturduk.

Evimize doğalgaz taktırmak istesek, sınıf için oluşturduğumuz fonksiyonu kullanırız.

```
class ev():  
  
    def __init__(self, metrekare, ısıtma, odaSayisi):  
        self.metrekare = metrekare  
        self.ısıtma = ısıtma  
        self.odaSayisi = odaSayisi  
  
    def dogalgaz_taktir(self):  
        self.ısıtma = "dogalgaz"
```

```
evim = ev(50, "klima", 2)
```

```
print("Evimin ısıtması", evim.ısıtma)
```

Evimin ısıtması klima

```
evim.dogalgaz_taktir()
```

```
print("Artık evimin ısıtması", evim.ısıtma)
```

Artık evimin ısıtması dogalgaz

Kalıtım | Python ile NYP

Bir şirketin çalışanlarını tasarlamak için sınıflar oluşturuyoruz. Bunun için muhasebeci, sekreter gibi sınıflar oluşturmamız gerekiyor. Fakat bu sınıfların hepsinin ortak metodları ve özellikleri bulunuyor. Tekrar tekrar özellikleri, sınıfları, metodları tanımlamak yerine tek bir çalışan sınıfı yaratalım ve diğerlerini bundan türetelim. **Kalıtım**'ın temel mantığı budur.

```
class calisan():  
  
    def __init__(self, isim, maas, departman):  
        print("Çalışan sınıfının init fonksiyonu")  
        self.isim = isim  
        self.maas = maas  
        self.departman = departman  
  
    def bilgileriogoster(self):  
        print("Çalışan sınıfının bilgileri.....")  
        print("İsim : {} \nMaas: {} \nDepartman: {}\n".format(self.isim, self.maas, self.departman))  
  
    def departman_degistir(self, yeni_departman):  
        print("Departman değişiyor....")  
        self.departman = yeni_departman
```

Kalıtım | Python ile NYP

Muhasebeci, sekreter ve müdür için oluşturacağımız sınıflar çalışan sınıfındaki tüm özelliklerle örtüştüğü için doğrudan çalışan sınıfından miras alındı. Böylelikle daha az efor harcadık ve daha düzenli, anlaşılır bir teknik oldu.

```
class muhasebeci(calisan): # Çalışan sınıfından miras alıyoruz.  
    pass # Pass Deyimi bir bloğu sonradan tanımlamak istediğimizde kullanılan bir deyimdir.
```

```
class sekreter(calisan): # Çalışan sınıfından miras alıyoruz.  
    pass # Pass Deyimi bir bloğu sonradan tanımlamak istediğimizde kullanılan bir deyimdir.
```

```
class mudur(calisan): # Çalışan sınıfından miras alıyoruz.  
    pass # Pass Deyimi bir bloğu sonradan tanımlamak istediğimizde kullanılan bir deyimdir.
```

```
muhasebeci1 = muhasebeci("Murat SAHIN",3000, "IK") # IK departmanında çalışan bir muhasebeci
```

Çalışan sınıfının init fonksiyonu

Ezme | Python ile NYP

Eğer biz miras aldığımız metodları aynı isimle kendi sınıfımızda tekrar tanımlarsak, artık metodu çağırdığımız zaman miras aldığımız değil kendi metodumuz çalışacaktır. Buna bir metodu **override** etmek denmektedir.

Örneğin artık Çalışan sınıfının init metodunu kullanmak yerine ekip_lideri sınıfında init metodunu override edebiliriz. Böylelikle ekip_lideri sınıfına ekstra özellikler ekleyebiliriz

```
class ekip_lideri(calisan):  
  
    def __init__(self, isim, maas, departman, ekip_kisi_sayisi): # Sorumlu olduğu kişi sayısı  
        print("Ekip liderinin init fonksiyonu")  
        self.isim = isim  
        self.maas = maas  
        self.departman = departman  
        self.ekip_kisi_sayisi = ekip_kisi_sayisi # Yeni eklenen özellik  
  
    def zam_yap(self, zam_miktari):  
        print("Maaşa zam yapılıyor....")  
        self.maas += zam_miktari  
        print("Maaşa zam yapıldı!")
```

Ezme ve Kalıtım | Python ile NYP

ekip_lideri sınıfı çalışan sınıfından **türetildi**.

Fakat çalışan sınıfında **zam_yap()** fonksiyonu bulunmuyor.

ekip_kisi_sayisi özelliği de çalışan sınıfına ait değil.

ekip_lideri sınıfı, türetildiği halde kendisine özel **zam_yap()** metodunu ve **ekip_kisi_sayisi** özelliğini tanımlamıştır.

```
ekip_lideri1 = ekip_lideri("Deniz KILINÇ", 5000, "IT", 10)
```

Ekip liderinin init fonksiyonu

```
ekip_lideri1.maas
```

5000

```
ekip_lideri1.zam_yap(200)
```

Maaşa zam yapılıyor....

```
ekip_lideri1.maas
```

5200

Üst Sınıf | Python ile NYP

super anahtar kelimesi özellikle ezdiğimiz bir metodun içinde aynı zamanda miras aldığımız bir metodu kullanmak istersek kullanılabilir. Yani **super**, miras aldığımız sınıfın metodlarını alt sınıflardan kullanmamızı sağlar.

```
class yonetici(calisan):  
  
    def __init__(self, isim, maas, departman, kisi_sayisi): # Sorumlu olduğu kişi sayısı  
        super().__init__(isim, maas, departman) # 3 tane özelliği Çalışan fonksiyonunun alıyor.  
        print("Yönetici sınıfının init fonksiyonu")  
        self.kisi_sayisi = kisi_sayisi # Ekstra özelliği de kendimiz yazıyoruz.  
  
    def bilgilerigoster(self):  
        print("Yönetici sınıfının bilgileri.....")  
        print("İsim : {} \nMaaş: {} \nDepartman: {} \nSorumlu olduğu kişi sayısı: {}".format(self.isim, self.maas, self.departman, self.kisi_sayisi))  
  
    def zam_yap(self, zam_miktari):  
        print("Maaşa zam yapılıyor....")  
        self.maas += zam_miktari
```

Ezme ve Kalıtım | Python ile NYP

Görüldüğü üzere **yönetici** sınıfından yönetici nesnesi oluşturulma esnasında 3 özellik **çalışan** sınıfından miras alınmıştır, “Çalışan sınıfının init fonksiyonu” mesajından bu durum teyit edilebilir.

Ayrıca yönetici sınıfının init fonksiyonu da çalışmıştır.

“**super.init(self, ...)**” kısmı önemlidir.

```
yoneticil = yönetici("Ali Kemal",10000,"Tüm Departmanlar",50)
```

```
Çalışan sınıfının init fonksiyonu  
Yönetici sınıfının init fonksiyonu
```

```
yoneticil.bilgilerigoster()
```

```
Yönetici sınıfının bilgileri.....  
İsim : Ali Kemal  
Maaş: 10000  
Departman: Tüm Departmanlar  
Sorumlu olduğu kişi sayısı: 50
```

Veri Bilimi Projelerinde Nesne Yönelimli Programlama?

- Projenizde *kod karmaşıklığı* gittikçe artıyor ve *soyutlama* ihtiyacı duyuyorsanız,
- Yazdığınız kodlar birer “science-script” olmaktan çıkıp servis koduna dönüşüyorsa,
- Bu kodların başka projeler içerisinde sadece sizin belirlediğiniz *erişim kurallarıyla* kullanılmasını istiyorsanız.

Evet, nesne yönelimli programlama kullanılmalıdır.

Ayrıca Python ile gerçekleştirdiğimiz neredeyse her veri bilimi projesinde, nesne yönelimli programlama yaklaşımı ile geliştirilmiş kütüphaneleri kullandığımızı da unutmamak gerekiyor.

Kaynakça

- [*https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century*](https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century)
- [*https://en.wikipedia.org/wiki/Data_science*](https://en.wikipedia.org/wiki/Data_science)
- [*https://www.yazilimbilimi.org/python-kalitim-inheritance/*](https://www.yazilimbilimi.org/python-kalitim-inheritance/)
- [*https://medium.com/deep-learning-turkiye/python-ile-veri-bilimine-dal%C4%B1%C5%9F-3f069260ebda*](https://medium.com/deep-learning-turkiye/python-ile-veri-bilimine-dal%C4%B1%C5%9F-3f069260ebda)