



20 may 2021

- aldığım eğitimler://

1. sitelerini:

1. w3school [python basics, numpy]
2. mobilhanem
3. guru99.com
4. zetcode
5. realpython [OOP]
6. geek4geeks [OOP]
7. codewars [8-7-6-5-4 kyu KATAs]
8. 100 helpful python tips
9. cleverprogrammer.com 100+ python mini projects
10. programiz 100 mini python projects
- 11.

2. kitaplar:

1. Sharvin Shah's Ultimate Guide to Python: How to Go from Beginner to Pro
2. python crash course 2nd edition
3. self-taught programmer - Cory Althoff [OOP]
4. fırat özgül Python - [OOP]
5. automate the basic stuff with python (180e kadar)
6. python notes for professionals

3. kurslar:

1. türkay ürkmez - 6 haftalık python'a giriş eğitimi
2. btk akademi python - [* - 22,23,24,25,26. levels]
3. python 3.10 - first look with claire burn

4. youtube:

1. learn python with freecodecamp in 4 hours - [functions, OOP]
2. tech with tim - [selenium, intermediate python]
3. Parvat Computer Technology - [assorted small projects]

5. muhtelif cheatsheets, PDFs, infographs, videos vs.

- python topics:

- | | | |
|----------------|------------------|-----------------|
| o data types | o exceptions | o GUI |
| o comments | o iterators | o XML |
| o strings | o scopes | o API |
| o variables | o file handling | o OCV |
| o tuples | o testing | o Kivy |
| o lists | o error handling | o Tkinter |
| o loops | o web scraping | o Django |
| o conditions | o selenium | o Bootstrap |
| o dictionaries | o numpy | o PyQt5 |
| o functions | o scipy | o SQL/NoSQL |
| o operators | o pandas | o BeautifulSoup |
| o modules | o matplotlib | o requests |
| o OOP | o UI/UX | o NLP (arastir) |

- some websites to exercise

1. exercism

2. hackerrank
3. programiz
4. codechef
5. zetcode
6. realpython
7. data flair
8. tutorials point
9. leetcode
- 10.

- some python libraries:

1. matplotlib: Matplotlib is by far the most common library in the Python community for exploration and data visualization. This library is the foundation of every other library. It provides countless charts and customization, from histograms to scatter plots, to customize and configure your plots, matplotlib sets down a variety of colors, themes, palettes, and other possibilities. If you are doing data analysis for a machine learning project or producing a report for stakeholders, matplotlib is certainly the most functional library.
2. Pygal: Pygal is an open-source python library that not only creates highly interactive plots but also creates SVG images of the graphs/plots so that we can use it and customize it accordingly. Pygal is highly customizable and creates graphs with a few lines of code. chartlar, grafikler, barlar yazmakta kullanılıyor. focuses on creating visualizations that work well on digital devices. You can use Pygal to emphasize and resize elements as the user interacts with your visualization, and you can easily resize the entire representation to fit on a tiny smartwatch or giant monitor.
3. numpy: NumPy is among the most powerful scientific computation Python libraries and is used extensively for Machine Learning and Deep Learning apps. NumPy is short for NUMerical PYthon. Complex computational machine learning algorithms need multidimensional array operations. NumPy shows solutions for large objects with multidimensional arrays and different tools to function with them. Features of NumPy
 1. It is an open-source Python library.
 2. It has matrix data structures and a multi-dimensional array.
 3. It can be used to conduct a range of mathematical functions on arrays.
 4. It is an extension of Numeric and numarray.
 5. It also has random number generators.
4. pandas: pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.
5. pygame: herhangi bir oyun yazmak için pyGame kullanılır.
6. django: Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel.
7. IRC: Internet Relay Chat is a text-based chat system. It enables discussions among any number of participants in so-called conversation channels, as well as discussions between only two partners – for example, in question-and-answer dialogues.
8. ocv: openCV resim işleme kütüphanesi. mesela face-mask detection programı yazmada kullanılır.
9. scipy: SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

- 10.scikit-learn: This is a Python library that is linked to NumPy and SciPy. Scikit-learn is known to be among the best libraries for dealing with complex data. In this library, there are a lot of modifications being made. The cross-validation function is one modification, offering the choice to use more than one metric. Few small changes have been made to many training approaches, such as logistics regression and nearest neighbors.
- 11.keras: Keras is known as being one of Python's finest machine learning libraries. It offers a simplified method for expressing neural networks. Keras also offers impressive utilities for compiling models, data-set analysis, graph visualization, and so much more. Keras utilizes either Theano or TensorFlow internally within the backend. It is also possible to use some of the many common neural networks, including CNTK. When we contrast it with other machine learning libraries, Keras is relatively sluggish. Since, by using back-end infrastructure, it generates a computational graph and then uses it to perform tasks.
- 12.pytorch: PyTorch is a massive library for machine learning that enables programmers to conduct GPU acceleration tensor computations, produce interactive computational graphs, and automatically calculate gradients. Other than that, PyTorch provides rich APIs to solve neural network-related application problems. The basis of this machine learning library is Torch, which is an open-source machine library built-in C with a wrapper in Lua. This machine library was released in Python in 2017, and the library has been getting popular and drawing a growing number of programmers of machine learning ever since its creation.
- 13.plotly: Plotly is a visualization library that is free and open-source. Developers love this library because of its top quality, publication-ready and immersive charts. A few instances of the charts that are available are Boxplot, heatmaps, bubble charts. Built on top of the D3.js, HTML, and CSS visualization library, it is one of the greatest data visualization tools accessible. It is developed using the Django framework and Python.
- 14.pyPdf2 : PyPDF2 allows manipulation of pdf in memory. This python library is capable of tasks such as:
 1. extract information about the document, such as title, author, etc.
 2. document division by page
 3. merge documents per page
 4. cropping pages
 5. merge multiple pages into one page
 6. encrypt and decrypt PDF files
 7. split pages and more
- 15.beautifulsoup: web scraping ile otomatik repetitive tasks yapmanı sağlar. There are so many libraries, frameworks, and tools that are used for the task of web scraping. Some of the most common libraries and modules in Python used for web scraping are:
 1. Scrapy
 2. Selenium
 3. BeautifulSoup
 4. Urllib.requestAfter scraping the data, the data is prepared so that it can be stored in a CSV file to create a dataset.

- some programming languages:

1. python - 2001 by Guido van Rossum. Python is a general-purpose, object-oriented, server-side interpreted, dynamic, multi-platform, high-level, open-source, non-compiled, scripting language. It uses the CPython Interpreter to compile the Python code to byte code.
2. java - 1995 by James Gosling. used to create applications in computers. original name was Oak. 360 derece turlar attığın siteler java ile yazılır.

3. C - 1972 by Dennis Ritchie. general purpose, imperative language. bad thing is C doesn't support OOP, that's why C++ invented. C biliyorsan bütün programlama dillerini temel düzey de olsa biliyordur. çoğunun fikir babası. OSlerin %90'ı C'de yazılıyor.
 4. C++ - 1983 by Bjarne Stroustrup. viewed by many as the best choice for large-scale projects. öğrenmesi çok zordur.
 5. javascript - 1995 by Netscape. JS code is written into an HTML page. Front end web işlerinde kullanılır.
 6. C# - 2000. windows uygulamaları, bankacılıkta vs. kullanılıyor. öğrenmesi zor.
 7. ruby - 1995. dynamic, OO, general-purpose programming language. one of the best to start with. ruby is a mix of LISP, SmallTalk, Ada, Perl, Eiffel languages
 8. php - server-side language designed for web development.
 9. objective-C - ios ve apple OS X'in ana dili. sadece ios geliştirmek için öğreniliyor.
- some usages of python
 - Databases (MySQL, Sqlite, MongoDB),
 - Veri Analizi (Numpy, Pandas),
 - Bot Yazımı (Selenium),
 - Restful Api Uygulamaları,
 - Masaüstü Uygulamaları (PyQt5),
 - Web Geliştirme (Django)
 - as a programmer you should try to write code as less as possible. someone may have already found out the solution that you would spend hours trying to solve.
 - a programming concept: dry - don't repeat yourself. do not write long, identical codes; try to combine them.
 - python'ın kullanma kılavuzu pep8'dir. daha iyi bir python programcısı olmak için bu kılavuzu mutlaka okumalısın.
 - IDE: interactive development environment.
<https://docs.google.com/spreadsheets/d/1l3x94P55qoxqYbq5GosWQ7IonZ4vR-4ZyCaImiVmCSk/pubhtml>
 - things you can do as a programmer:
 1. General Programming:
 1. While there are many who might consider themselves to be generalist programmers, there aren't many jobs for generalists. People usually get hired for more specific roles, such as web development or data science work.
 2. Web Development:
 1. Web development is often broken into front end work and back end work. Front end developers focus on the interface that users see in the browser. They try to make sites look appealing on the full range of devices available today, and they try to make sites respond quickly and smoothly to user input. Back end developers focus on data management and server-side processing. This usually involves a strong understanding of working with databases and APIs. Full-stack developers work on the front end and the back end, and can build full working prototypes of web sites independently. You can find work building new sites, or maintaining any of the millions of existing interactive sites. You can work on small development teams, or you can be part of a large corporation responsible for some of the largest and most popular sites on the internet. There's also a lot of web development work that's done entirely on internal networks, for apps that are used exclusively within an organization.
 3. App Development:
 1. If you want to build iOS apps, you'll be using Swift and XCode. If you want to build Android apps, you'll be using Java or related languages like Kotlin. You're not wasting your time learning Python, though. For one thing, Python is a much friendlier first language than Swift. The fundamentals of programming

you learn through Python will serve you well when you start digging into other languages like Swift, Java, or Kotlin.

4. Data Science:

1. Data scientists collect, clean, analyze, and visualize data. requires elevated mathematic skills.

5. Machine Learning:

1. Machine Learning is an advanced case of applied statistics and data analysis. If you want to work on ML algorithms, you'll need a really strong background in math.

6. Artificial Intelligence:

1. Much of what is sold as artificial intelligence is really just applied statistics, or Machine Learning. True AI, or general AI, is still a far-off goal.

7. Business Software Development:

1. Almost every business in operation has some need for software development.

8. Game Development:

1. it may not be as enjoyable and lucrative as you might think it is.

9. SaaS:

1. SaaS stands for Software as a Service. This is what often takes the place of desktop apps on current devices. Instead of building a piece of software that users install on their computers, you build the functionality on a server, and you provide a way for users to interact with that software - either through a browser, or through an app. You run the software on a server, and you offer this as a service to end users.

10. Freelance Work: you can either work as a freelance on your own or with a freelancing agency.

11. System Administration:

1. System Administration has often been seen as separate from programming. But there is so much automation involved in sysadmin work, that many system administrators end up learning to program at some point. If you're just starting out in the technical world, you might find that you really like sysadmin work. You'd be responsible for building and maintaining technical infrastructure. For example, you might work on a SaaS project, where you'd be responsible for the deployment infrastructure. Or you might work at a large company where you'd be responsible for maintaining the organization's network, and much of the hardware on the network.

12. Embedded Software:

1. yazıcı programı yazmak vs.

13. Domain-specific Software:

1. Domain-specific software refers to software that's written to solve a problem in a specific domain, or field of work. Some examples include scientific software, medical software, and real estate software.

14. Robotics:

1. People have been writing software for robotic applications for a long time, and this work is only getting more interesting. If you're interested in working with robotics, you can focus on low-level code that makes efficient use of physical devices, or you can work on higher-level code that aims to solve real-world challenges.

15. Financial Trading:

1. If you have really strong math skills and love money, you may want to get into financial trading software, also known as algorithmic trading. This kind of work tends to be high-stress, but if you do it well it can pay really well. There are no shortcuts in this field; if you think you've thought of a way to

game the system through code, then what you're thinking of is probably illegal, or will likely lose all of your money faster than you can imagine.

- IDLE: Interactive Development Environment
- some writing techniques
 - o camelCaseTechniqueWriting
 - o PascalTechniqueWriting
 - o kebab-technique-writing
 - o snake_technique_writing
- One of its most visible features is that python does not use semicolons nor brackets; it uses indentation instead.
- There are three main implementations of Python:
 1. CPython: CPython is implemented in C language. It is the most widely used implementation of Python. When people talk about Python language, they mostly mean CPython.
 2. IronPython: IronPython is implemented in C#. It is part of the .NET framework.
 3. Jython: similarly, Jython is an implementation of the Python language in Java. Jython program is translated into the Java bytecode and executed by the JVM (Java Virtual Machine).
- Python scripts have .py extension.
- import this yazdığında Zen of Python şiiiri yazdırılır.
- comment'lerin başında # bulunur. docstringler ise """ içine yazılır """. buna yorum satırı denir. C'de // ile kullanılır.
- shift'e basılı tutup sağ sol oklarla tek tek seçebilirsiniz, ctrl + shift'le komple seçebilirsiniz.
- Python2 ile yazılmış bir program Python3'te çalışmaz. Aynı şekilde Python3 ile yazdığınız bir program Python2'de çalışmaz.
- Pythonistas, as members of the community call themselves, meet around the world in thousands at PyCon conferences.
- python sürümünü öğrenmek için cmd'ye python --version yaz.
- youtube, spotify, instagram, reddit vs. python'la yazılmıştır
- Many Python programmers recommend that each line should be less than 80 characters.
- pyCharm'da birden çok satıra aynı anda bir şeyler yazmak istiyosan shift + alt basılı tutup fareyle satırları seçmen gerekir:

```
print("Water = 100ml")
print("Milk = 50ml")
print("Coffee = 76ml")
print("Money = $5")
```

- fonksiyonların hemen altına """ içinde """ tanım yazarsan daha sonra çağırdığın fonksiyonun üzerine geldiğinde bu tanıma hatırlatır ve kolaylık olur.
- emoji klavyesi windows + .
-

#introduction

- python'da 9 temel veri tipi vardır
 1. strings

```
"hello"
```

stringler birden çok karakterden oluşan " " ya da ' ' aralığındaki her şeydir. "500" de bir stringdir "hello, world" da.

2. integer

```
300
```

ondalıklı olmayan pozitif ya da negatif tüm sayılardır. C'nin aksine python3.0'da max ya da min integer kavramı yoktur otomatik olarak long'a geçer büyüklük sınırını aştığında.

3. lists

```
a = [1, 2, 3, "bugra", "kara"]
```

[] içine yazılan, elemanları virgüllerle ayrılan veri tipidir. bu elemanlar farklı veri tiplerinde olabilir.

4. dictionaries

```
a = {"a":1, "b":2, "c":3}
```

{ } içine yazılan, elemanlarının (key) değerlerle eşleştirildiği (value) veri tipidir. virgüllerle ayrılır.

5. tuple

```
a = ("bugra", "kara", 1, 2, True)
```

liste gibidir fakat parantez içine yazılır.

6. float

```
10.2
```

ondalıklı tüm sayılardır.

7. long integer

```
230e
```

8. complex

```
3 + 4j
```

9. sets

```
a = {"a", "b", 10.5, False}
```

built-in data types include False, True, and, or, None vs. None is the value for the absence of any value. it is always smaller than any number, including negatives.

- you can convert data types to one another using:

- o int()
- o float()
- o str()
- o bool()
- o list()
- o set()
- o tuple()

- my first code:

```
print("hoşçakal dünya")
```

- help() metodu bir keywordün ne işe yaradığını öğrenmek için kullanılır:

```
print(help(len))
```

- dir() metodu bir modülün bütün metotlarını verir

- min() ve max() metotları bir intejerin en yüksek ve en düşük değeri verir:

```
x = min(5, 10, 25)
y = max(5, 10, 25)
print(x)
print(y) >>> 5 25
```

- abs() method gives the absolute power of any given number:

```
x = abs(-7.25)
print(x) >>> 7.25
```

- if you want to extend the list of mathematical functions, you may use math module:

```
import math
```

- math.sqrt() method gives the square root of a number:

```
import math
a = math.sqrt(16)
print(a) >>> 4.0
```

- math.ceil() aşağı math.floor() yukarı yuvarlar:

```
import math
x = math.ceil(1.4)
y = math.floor(1.4)
print(x) >>> 2
print(y) >>> 1
```

- `math.pi` metodu `pi`'nin değerini yazdırır:

```
import math
x = math.pi
print(x) >>> 3.141592653589793
```

- `value` type'larla uğraşırken listenin kendisiyle, `reference` type'larla uğraşırken listenin adresiyle uğraşıyoruz. esas dosya, kısayol dosya gibi bir ayrım düşünebilirsiniz.
- terminalde
 - o `cls` ya da `clear` ekranı temizler.
 - o `mkdir xxx` yazarsan o anki klasörde `xxx` adlı bir alt klasör yaratırsın.
 - o `rmdir` dosyadaki `xxx` adlı klasörü siler.
 - o `ls` klasörleri listeler.
- A literal is any notation for representing a value in a Python source code. Technically, a literal is assigned a value at compile time, while a variable is assigned at runtime.

```
age (literal) = 29 (value)
nationality (literal) = "Hungarian" (value)
```

Here we assign two literals to two variables; number 29 and string "Hungarian" are literals.

- operators:
 - o Arithmetic operators: (+) , (-), (*), (/), (**) üs alma, (%) sadece kalan alma, (//) tam bölen alma
 - o Assignment operators: = , += , -= vs.
 - o Comparison operators: ==, !=, >= vs.
 - o Logical operators: and, or, not
 - o Identity operators: is, is not
 - o Membership operators: in, not in
 - o Bitwise operators: &

Augmented assignment statement	Equivalent assignment statement
<code>spam += 1</code>	<code>spam = spam + 1</code>
<code>spam -= 1</code>	<code>spam = spam - 1</code>
<code>spam *= 1</code>	<code>spam = spam * 1</code>
<code>spam /= 1</code>	<code>spam = spam / 1</code>
<code>spam %= 1</code>	<code>spam = spam % 1</code>

- Üs almak için `x ** 2` dersin. Üs almanın bir diğer yolu da `pow()` metodudur:

```
print(pow(2,5))
```

karekök almak için `** 0.5` yaparsın. bir sayının 0.5. kuvveti o sayının kareköküdür.

- kod satırları rakam ile başlayamaz. `number1` olabilir ama `1number` olamaz.
- genelde üç çeşit hata vardır:
 - o programcı hatası (error): typo
 - o mantık hatası (bug): program çalışır ama hata vermez, istediğin gibi de çalışmaz. en zor hata türü
 - o istisnalar (exception): i.e sifıra bölme, 0'ın 0. kuvvetini alma vs.
- `float`'la `integer`'ın toplamı floattır sonu küsuratlı olmasa dahi.

VARIABLES

- = işareti normalde eşitlik anlamına gelirken programlama dillerinde atama operatörüdür, sağdakinin değeri soldaki olsun demektir. eşitlik/denklik ise `=` ile ifade edilir. birincisine karar verebilirken ikincisine veremeyiz.
- variables are case sensitive. that is `liste1` is not same with `Liste1`. aynı şekilde `age` `Age` farklıdır.
- TR karakter kullanmamaya çalış.
- variables:
 - o sayıyla başlamaz (`1_liste`)

- o underscore haricinde özel sembol içermez (% * = +)
- o aralarında boşluk olmaz
- o türkçe karakter olabilir ama risklidir
- o keyword'ler isim olarak verilemez (and or while True vs.)
- değişken tanımladıktan sonra del değişkenİsmi diyerek o tanımlamayı silebilirsin.
- değişken ismi olarak özel terimleri (True, False, and vs.) kullanamazsın.
- python'da özel terimleri (keywords) görmek için:


```
import keyword
print(keyword.kwlist)
```
- input'tan sonra gelen satır çok uzunsa \ kullanarak ayırabilirsin:


```
print("bu satır çok uzun olduğu için \ kullanarak " + \
      "ayırarak zorunda kaldık.")
```
- çoklu atama:


```
values = 10, 20, 30
x, y, z = values
print(x, y, z) >>> 10 20 30
```

eğer values değeri, eşleştirdiğin harflerden çok sayıdaysa şöyle yapabilirsin:

```
values = 10, 20, 30, 40, 50, 60
x, y, *z = values #başına yıldız koyduğun için
print(x, y, z) >>> 10 20 [30, 40, 50, 60] 10u alır 20yi alır z'yi liste olarak alır.
```
- here are the two ways of swapping variables in python (değerlerini birbirine atama):


```
"""birinci yol üçüncü bir geçici değişken tanımlamak"""
a = 5
b = 10

c = a
a = b
b = c

print(f"a = {a}") >>> 10
print(f"b = {b}") >>> 5

"""ikinci yol virgülle ayırarak atama"""
a = 3
b = 5

a, b = b, a

print(f"a = {a}") >>> 5
print(f"b = {b}") >>> 3
```

STRINGS

- cardinal string methods:
 - o capitalize()
 - o center()
 - o casefold()
 - o count()
 - o endswith()
 - o startswith()
 - o encode()
 - o find()
 - o format()
 - o index()

- o isalpha()
 - o isnum()
 - o join()
 - o just()
 - o lower()
 - o upper()
 - o swapcase()
 - o strip()
 - o partition()
 - o translate()
 - o replace()
 - o find()
 - o split()
 - o title()
 - o zfill()
- bir string rakamlardan değil harflerden oluşan veri tipidir ve ' ' ya da " " içinde yazılır. bunların içinde olduğu müddetçe rakam, sembol vs. her şey string farzedilir:

```
myStr = "bugra123+%&"
```
 - stringin içine \n yazarsan bir alt satıra geçirir.

```
a = "benim \nadim"
print(a) >>>
benim
adim
```
 - \t yazarsan bir tab boşluk (8 boşluk) bırakır:

```
a = "merhaba \t benim"
print(a) >>> merhaba          benim
```

iki kelime arasına boşluk koymazsan \t işe yaramaz
 - \' : hata verdirmeden tek tırnak yazdırır, \" hata verdirmeden çift tırnak yazdırır:

```
print("\"merhaba\"") >>> "merhaba"
```
 - \b yazarsan bir önceki string'i siler (yutar):

```
print("Python\bba\bs") >>> Pythos
```

hiç yazmasak daha mantıklı ı
 - tırnağın başına * atarsan stringi harflere böler:

```
print(*"merhaba") >>> m e r h a b a
```

bunu sep=" " metoduyla da zenginleştirebilirsin:

```
print(*"bugrahan",sep="+") >>> b+u+g+r+a+h+a+n
```
 - string'in başına r yazarsan bunları elimine eder:

```
print(r"Another world\nanother cray\bo\tn") >>> another world another person
```
 - Strings are immutable. This means that once defined, they cannot be changed. Many Python methods, such as replace(), join(), or split() modify strings. However, they do not modify the original string. They create a copy of a string which they modify and return to the caller. karakter dizileri aslında değiştirilemez veri tipleri olduğu için gördüğümüz bu değişiklikler aslında gerçek değildir (reference değişiklikleridir, adres değil).
 - python'da yorum satırları """ abc """ ya da #abc olarak yazılır
Python strings can be created with single quotes, double quotes, or triple quotes. When we use triple quotes, strings can span several lines without using the escape character. other than that you have to write your comment in one single line:

```
a = """
benim
adim
"""
print(a)
```
 - string slicing:

```
a = a[:10]    baştan 10. indekse kadar
a = a[3:7]    3. indeksten 7. indekse kadar
a = a[::-1]   sondan başa doğru tersten
a = a[4:20:3] 4. indeksten 20. indekse kadar 3'er 3'er
a = a[2::15]  2. indexten 15. indexe 3'er 3'er yazdırır
```

- eski sürüm pythonlarda long isminde bir data type daha vardı fakat sonradan int ile birleştirildi.

- girilen iki sayıyı toplama:

```
sayi1 = float(input("sayi1'i giriniz: "))
sayi2 = float(input("sayi2'yi giriniz: "))
toplam = float(sayi1) + float(sayi2)
print(toplam)
```

- girilen 2 Sayının ortalamasını bulma:

```
sayi1 = float(input("sayi1'i giriniz: "))
sayi2 = float(input("sayi2'yi giriniz: "))
ortalama = (float(sayi1) + float(sayi2)) / 2
print(ortalama)
```

- format() metoduyla da bunu yapabilirsin:

```
sayi1 = float(input("sayi1'i giriniz: "))
sayi2 = float(input("sayi2'yi giriniz: "))
toplam = float(sayi1) + float(sayi2)
print("toplam: {0}".format(toplam))
```

format metodunda stringin içine boş da olsa { } açman gerekir ki format metodu çalışsın.

- % metodunun kullanımı:

```
print("there are %d cars in the parking lot" %15) >>> there are 15 cars in the parking lot
%d dersin integer ya da float girmek zorundasın. %s dersin de string. string " " içinde olmalı:
```

```
print("his name was %s" %"nicholson")
```

%a da kullanabilirsin:

```
print("there are %a cars in the parking lot" % "on bes")
```

eğer birden fazla gireceksen virgülle ayırman gerekir:

```
print("there are %d cars and they're %a" % (15, "red")) >>> there are 15 cars and they're red
```

bir değişken de kullanabilirsin:

```
a = 20
```

```
print("bugün %s yaşıma girdim." %a) >>> bugün 20 yaşıma girdim.
```

farklı kullanımları:

- o %d - integer
- o %s - string
- o %f - float

%s multiple usage örnek:

```
a = "bugra"
b = "ugur"
c = "cemre"
print("1.sinin adı %s, 2.sinin adı %s, 3.sünün adı %s" % (a, b, c))
```

- inputu şöyle de kullanabilirsin:

```
print("enter:")
b = input()
print(f"you've entered {b}")
```

- binary'ye karşılık gelen integeri yazdırma:

```
print("{:b}".format(1)) >>> 1
print("{:b}".format(2)) >>> 10
print("{:b}".format(10)) >>> 1010
```

- f string de bir string yazdırma metodudur. format ile f string birbirine benzer:

```
print("Toplam: {}".format(toplam))
```

gives the same result with

```
print(f"toplam: {toplam}")
```

örnek:

```
a = "klavye"
```

```
b = "oyuncak"
```

```
print("teknosadan {0} toyshop's'tan da {1} aldım".format(a,b))
```

bunun f string ile yazılmış hâli:

```
a = "klavye"
```

```
b = "oyuncak"
```

```
print(f"teknosadan {a} toyshop's'tan da {b} aldım.")
```

değişken tanımlayan f string kullanırken parantez kullanmak zorunda değilsin:

```
fav_num = 42
```

```
msg = f"My favorite number is {fav_num}."
```

```
print(msg) >>> My favorite number is 42.
```

fakat bu özelliği printle kullanamazsın.

```
dil = "Python"
```

```
zorluk = "30/100"
```

```
print f"öğrenmek istediğim dil {dil} ve bu dilin zorluğu {zorluk}." >>> ERROR
```

örnek2:

```
Ülkeler = ["Türkiye", "Ukrayna", "ABD"]
```

```
mesaj = f"yaşadığım ülke {Ülkeler[2].lower()}"
```

```
print(mesaj) >>> yaşadığım ülke abd
```

f string + for loop kullanımı:

```
countries = ["Russia", "America", "Austria", "Switzerland"]
```

```
for i in countries:
```

```
    mesaj = f'I want to see {i} in the future'
```

```
    print(mesaj) >>>
```

```
I want to see Russia in the future
```

```
I want to see America in the future
```

```
I want to see Austria in the future
```

```
I want to see Switzerland in the future
```

eğer print'i indented yazmazsan sadece son elemanı yazdırıyor.

örnek2:

```
gelenler = ["ali", "ahmet", "ayşe", "mehmet", "fatma"]
```

```
for i in gelenler:
```

```
    mesaj = f"pikniğe geldi: {i}"
```

```
    print(mesaj)
```

aynısını formatla da yapabilirsin:

```
gelenler = ["nick", "jessica", "brian", "amanda"]
```

```
for i in gelenler:
```

```
    print("pikniğe gelenler".format(i))
```

f string + input + condition kullanımı:

```
tables_ordered = int(input("how many tables do you want? :"))
```

```
available_tables = 25
```

```
if tables_ordered < available_tables:
```

```
    print("we have enough tables")
```

```
elif tables_ordered > available_tables:
```

```
    print(f"sorry, we have only {available_tables} tables. you'll need
```

```
{tables_ordered-available_tables} more")
```

- though you can use + symbol to print line by line the same data types, you cannot directly combine strings and integers like that:

```
age = 43
txt = "my age is" + age
print(txt) >>> Error
```

- format metodu kullanarak bu sorunu aşabilirsiniz:

```
name = "bugra"
surname = "kara"
print("benim adım { } and my surname is { }".format(name,surname)) >>>
benim adım bugra and my surname is kara
```

{ } ler yazdığın sıraya göre otomatik yerleştirir. Yerlerini değiştirmek için { } lere sırasıyla 1 ve 0 yazarsan bugra ve kara yer değiştirir.

```
name = "jeremy"
surname = "irons"
print("My name is {1} {0}. I live in Kentucky.".format(name, surname)) >>>
My name is irons jeremy. I live in Kentucky.
```

Sıralarıyla oynamak için .format'tan sonra değişkenleri harflendirip o harfleri { } ler içine de yazabilirsin.

```
name = "daniel"
surname = "cosway"
print("My name is {b} {a}. I live in Kentucky.".format(a = name, b = surname)) >>>
My name is cosway daniel. I live in Kentucky.
```

fakat sadece harf verebilirsin, integer değil.

format metodunun mantığı şöyle:

```
a = "aa"
b = "bb"
c = "cc"
text = "I want { }, { }, and { }"
print(text.format(a,b,c))
```

- join() metodu bir stringde harf aralarına istediğiniz stringi ekler.
"istediğin bir şey".join(mesaj) şeklinde.

```
mesaj = "MERHABA benim Adım"
mesaj = "*".join(mesaj)
print(mesaj) >>> M*E*R*H*A*B*A* *b*e*n*i*m* *A*d*i*m
```

örnek

```
a = ["harley", "suzuki", "kmt", "ducati"]
b = "--".join(a)
print(b) >>> harley--suzuki--kmt--ducati
```

örnek2:

```
a = "4124"
a = "+".join(a)
print(a) >>> 4+1+2+4
```

- partition() method is another method which can be used for splitting strings. It will split the string at the first occurrence of the separator and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator:

```
a = "harley davidson is founded in 1903 in USA"
b = a.partition("founded")
print(b) >>> ('harley davidson is ', 'founded', ' in 1903 in USA')
```

- len() metoduyla bir string'deki karakter sayısını öğrenebilirsin.

```
a = "harley davidson is founded in 1903 in USA"
print(len(a)) >>> 41
```

- upper() metodu bütün karakterleri büyük harfe çevirir:

```
mesaj = "merhaba benim adım"
mesaj = mesaj.upper()
print(mesaj) >>> MERHABA BENİM ADİM
```

- lower() metodu bütün karakterleri küçük harfe çevirir:

```
mesaj = "MERHABA benim Adım"
mesaj = mesaj.lower()
print(mesaj) >>> merhaba benim adim
```

- title() metodu her kelimenin baş harfini büyük harfe çevirir

```
mesaj = "MERHABA benim Adım"
mesaj = mesaj.title()
print(mesaj) >>> Merhaba Benim Adim
```

- ve bir ayrıntı da kelimedeki sadece baş harfi büyük yapar, diğer harfler de büyükse küçüğe çevirir:

```
a = "bBugGRA"
a = a.title()
print(a) >>> Bbuggra
```

- capitalize() sadece cümlenin ilk harfini büyük harfe çevirir.

```
a = "merhaba dünya"
a = a.capitalize()
print(a) >>> Merhaba dünya
```

- swapcase() metodu büyük harfi küçüğe küçüğü büyüğe çevirir:

```
a = "BeniM ADiM"
a = a.swapcase()
print(a) >>> bENiM adIM
```

küçüğü büyük, büyüğü küçük harf yapan program:

```
def to_alternating_case(string):
    return "".join([c.swapcase() for c in string])
```

yukarıdaki

```
return "".join()
```

metodu çok kullanılır. önce boş bir string tanımlayıp daha sonra join metodu ile o stringe istediğin şeyleri eklersin.

- örnek:

```
def func(n):
    return f"{n} plays banjo" if n.startswith("R") or n.startswith("r") else \
    f"{n} does not play banjo"
```

- zfill() metodu ile karakter dizimizi karakter belirttiğimiz karakter sayısından ne kadar az ise başına o kadar sıfır ekler:

```
a = "BeniM ADiM" #10 karakter
a = a.zfill(13)
print(a) >>> 000BeniM ADiM
```

- startswith() metodu bir karakterin belirlediğimiz karakterle başlayıp başlamadığını sorar:

```
a = "bilgisayar"
print(a.startswith("a")) >>> False
```

endswith() de tam tersi.

- strip() metodu baş ve sondaki boşlukları siler. sadece soldan silmek için .lstrip() , sağdan silmek için .rstrip().

```
a = "    merhaba    "
a = a.strip()
print(a) >>> |merhaba|
```

neyi silmek istiyosan onu yazabilirsin boşluğa:

```
a = "?python?"
a = a.strip("?")
```

fakat bu metot stringin içindeki işaretleri kaldırmaz. bunun için replace() metodunu kullanabilirsin:

```
a = "?p?yt?ho?n?"
a = a.replace("?", "")
```

- `split()` metodu cümleyi her boşluktan itibaren stringe çevirir ve liste halinde verir. `Split()` metodunda `()` içine (tırnak içinde olacak şekilde) ne yazarsanız ondan itibaren parçalar verdiğiniz cümleyi. `a = a.split(".")` gibi.

```
a = "merhaba arkadaşlar bu cümleyi liste olarak yazdıracağız"
a = a.split()
print(a) >>> ['merhaba', 'arkadaşlar', 'bu', 'cümleyi', 'liste', 'olarak', 'yazdıracağız']
örnek2:
```

```
a = "benim adım, bugra, kar,a "
a = a.split(",")
print(a) >>> ['benim adım', ' bugra', ' kar', 'a ']
```

- `find()` metodu bir stringin içinde aradığın şeyin kaçınca index'te olduğunu yazdırır, yoksa -1 yazdırır. kelime aratırsan kelimenin (bulursa) ilk harfinin başladığı index'i yazdırır.

```
mesaj = "1234567890"
cevap = mesaj.find("0")
print(cevap) >>> 9
```

`rfind()` metodu sağdan `lfind()` metodu soldan aramaya başlar.

- stringdeki küçük harf sayısını veren program:

```
def lower(n):
    a = 0
    for i in n:
        if i.islower():
            a += 1
    return a
```

daha kısa yolu:

```
def lowercase_count(strng):
    return sum(a.islower() for a in strng)
```

niye çünkü `islower()` doğruysa `true` yani 1 döndürecek 1leri `sum()` ile toplayıp cevabı buluyor

- yılın hangi yüzyılda olduğunu veren program:

```
def century(year):
    return (year + 99) // 100
```

- `replace()` metodu string içindeki şeyleri başka şeylerle değiştirmek için kullanılır:

```
mesaj = "python ogrenmek cok zor"
mesaj = mesaj.replace("o", "0")
print(mesaj) >>> pyth0n 0grenmek c0k z0r
```

- `center()` metodu stringi yazdığın karakter alanına ortalar:

```
mesaj = "python ogrenmek cok zor"
mesaj = mesaj.center(50, "*")
print(mesaj) >>> *****python ogrenmek cok zor*****
```

yıldızdan önce yazdığın sayı `len(mesaj)`dan büyük olmalıdır. 2 fazla olursa `*mesaj*` yazdırır.

- `rjust()` ve `ljust()` metodu sağa ya da sola yazdırır istediğin sembolü.

```
a = "bugra".ljust(10, "*")
print(a) >>> bugra*****
a = "bugra".rjust(10, "*")
print(a) >>> *****bugra
```

- `count()` metoduyla bir harfin ya da rakamın bir string'de kaç kere olduğunu saydırabiliriz:

```
course = "Python Kursu: Baştan Sona Python Programlama Rehberiniz (40 saat)"
result = course.count("a")
print(result) >>> 8 kere a varmış.
```

belli bir aralıkta aratmak için `.count("a", 0, 10) >>> 0'la 10. indeks arasında kaç tane "a" var?`

örnek:

```
a = "3242363465475980985082347598347598275982758392657823645897235"
```

```
print(a.count(str(4))) >>> 6
```

- isalpha() metodu bir string ifadenin harflerinin alfabetik olup olmadığını sorgular:

```
a = "a b c e d"
```

```
print(a.isalpha()) >>> False
```

ya da

```
a = "abcd".isalpha()
```

```
print(a) >>> True
```

isdigit() metodu rakamatik olup olmadığını sorgular.

```
a = "123"
```

```
a = a.isdigit()
```

```
print(a) >>> True
```

isinstance(a, b) metodu, a'nın b'nin bir örneği olup olmadığını sorar:

```
a = False
```

```
print(isinstance(a, bool)) >>> True
```

ya da

```
a = 166.4
```

```
print(isinstance(a, float)) >>> True
```

- True ve False değerleri sayısal olarak da 1 ve 0'a eşittir:

```
print(True+True) >>> 2
```

- translate() metodu stringdeki harfleri istediğin şeyle değiştirir. mesela bir stringdeki A'ları T, T'leri A, C'leri G, G'leri de C yapacaksın:

```
def dna_swap(dna):
```

```
    return dna.translate(string.maketrans("ATCG","TAGC"))
```

- bir stringdeki boşlukları kaldırma:

```
def func(n):
```

```
    return n.replace(" ","")
```

- bir ismin sadece baş harflerini yazdıran program:

```
def func(n):
```

```
    return ".".join([i[0].upper() for i in n.split()])
```

```
print(func("Bugra Kara")) >>> B.K
```

- girilen sayı kadar '10' çifti yazdıran program:

```
def func(n):
```

```
    return ("10"*n)[:n]
```

```
print(func(4)) >>> 1010
```

- bir stringin başındaki ve sonundaki harfi silen program:

```
def func(n):
```

```
    return n[1:-1]
```

```
print(func("bugra")) >>> ugr
```

- stringin her bir kelimesini array'e dönüştüren program:

```
def func(n):
```

```
    return [i for i in n.split(" ")]
```

```
print(func("bugra kara")) >>> ["bugra", "kara"]
```

ikinci yolu

```
def func(n):
```

```
    return n.split() if len(n) > 0 else [""]
```

- girilen sayıyı tersten arraye dönüştüren program:

```
def func(n):
```

```
    return sorted(int(i) for i in str(n))[::-1]
```

```
print(func(123456)) >>> [6, 5, 4, 3, 2, 1]
```

- bir stringdeki harfleri spesifik başka harflerle değiştirme:

```
def func(string):
```

```
    correction = {"0":"0", "5":"S", "1":"I", "3":"E"}
```

```
    for i in string:
```



```

        if i in correction:
            string = string.replace(i, correction[i])
        return string
print(func("B0YL3 G3LMİ5 B0YL3 G1D3R")) >>> BÖYLE GELMİS BÖYLE GİDER

```

bir başka yolu da maketrans() metodudur:

```

def func(string):
    return string.translate(str.maketrans("0315","0EIS"))

```

bir başka yolu da replace() metodudur:

```

def func(string):
    return string.replace("5","S").replace("1","I").replace("3","E").replace("0","O")

```

- sep() metodu virgülle ayırdığın elemanların arasına gelecek olan şeyi belirlemeni sağlar:

```

print("bugra", "kara", "molla", sep = "*") >>> bugra*kara*molla

```

end() metodu da parametrenin sonuna neyin geleceğine karar vermeni sağlar:

```

print("bugra", "kara", "molla", end = "*") >>> bugra kara molla*

```

end() metoduyla sep() metodunun arasındaki fark sep sadece aralarına istediğin stringi getirir, sonuna değil.

```

print("1", "2", "3", "4", sep = " mumdur ") >>> 1 mumdur 2 mumdur 3 mumdur 4 mumdur

```

sep = "\n" yazarak kelimeleri alt alta da yazdırabilirsin.

örnek:

```

def func(n):
    count = ""
    for i in range(1,n+1):
        count = count + "{ } sheep..".format(i)
    return count
print(func(3)) >>> 1 sheep..2 sheep..3 sheep..

```

- input() operatörü kendinden printlidir. yani başına print(input()) yazmana gerek yoktur.
- çoklu variable atama:

```

one, two, three, four = 1, 2, 3, 4

```

- 3 adet stringin harflerini birer birer peşpeşe yazdırma:

```

def triple_trouble(one, two, three):
    return "".join([one[i] + two[i] + three[i] for i in range(len(one))])
print(triple_trouble("aa","bb","cc")) >>> abcabc

```

-

#list

- list methods:

- o index()
- o append()
- o extend()
- o insert()
- o remove()
- o count()
- o pop()
- o reverse()
- o sort()
- o copy()
- o clear()
- o zip()
- o del
- o all()
- o any()

- liste: köşeli parantezler içine alınan, virgüllerle ayrılan kümelere denir. öğeleri string, integer, float vs. olabilir.

```

my_list = [122, "bugra", 10.4, True]

```

- bir listede silme:
 - istediğin stringi silmek için → `remove("silinecek_kelime")`
 - istediğin indeksi silmek için → `pop(indeks)`
 - komple silmek için → `del liste`
 - bazı elemanlarını silmek için `use del + slicing` → `del liste[::3]` (her 3 elemandan birini siler)
 - içini boşaltmak için `liste.clear()`
- liste ve string farkı:

```
a = "merhaba"
b = "merhaba"
c = a is b
print(c) >>> True
```

listelerde ise böyle değil:

```
a = ["merhaba"]
b = ["merhaba"]
c = a is b
print(c) >>> False
```

- in operatörü:

```
x = ["harley", "davidson"]
print("davidson" in x) >>> True
```

not in operatörü:

```
x = ["harley", "davidson"]
print("yamaha" not in x) >>> True
```

- liste eleman atama:

```
list = [1,2,3,4]
one, two, three, four = list
print(two) >>> 2
```

- `heapq.nlargest/smallest` metodu ile bir listeden x kadar en büyük ya da küçük eleman alma:

```
import heapq
liste = [12, 5, 315, 3, 100, 32, -50, 19, 60, 60.5]
print(heapq.nlargest(3, liste)) >>> [315, 100, 60.5]
```

ve

```
import heapq
liste = [12, 5, 315, 3, 100, 32, -50, 19, 60, 60.5]
print(heapq.nsmallest(3, liste)) >>> [-50, 3, 5]
```

- `*list` metodu ile we can extract all elements of a list:

```
list = ["a", 1, 2, 3]
print(*list) >>> a 1 2 3
```

[list] * int metodu ile bir listenin elemanlarını arttırabilirsin:

```
a = ["merhaba"]
print(a*3) >>> ['merhaba', 'merhaba', 'merhaba']
```

- range metodu:

```
for i in range(1,11):
    print(i)
```

liste içinde range metodunu kullanma:

```
n1 = ["a" for i in range(15)]
n2 = ["a"] * 15
print(n1) >>> ['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']
n1[0:10] = [10] * 10
print(n1) >>> [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 'a', 'a', 'a', 'a', 'a']
```

örnek:

```
liste_range = ["i" for i in range(5)]
liste_range[:3] = "b" * 3
print(liste_range) >>> ['b', 'b', 'b', 'i', 'i']
```

range() metoduyla bir liste oluşturmak için:

```
sayılar = list(range(10, 21))
```

```
print(sayılar) >>> [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

range metodunda geriye doğru 3'er 3'er yazdırma:

```
for i in range(20, 1, -3):
```

```
    print(i) >>> 20den 1e kadar 3'er 3'er azaltarak yazdırır
```

1'er 1'er saydırsın istiyorsan -1:

```
def func(n):
```

```
    return [i for i in range(n, 0, -1)]
```

bir başka yolu:

```
def func(n):
```

```
    return sorted([i for i in range(1, n+1)], reverse=True)
```

range metodu, for döngüsünün bir türüdür. range(a, b) a'dan b'ye kadar (a dahil b değil) yazdırır:

```
for i in range(4,10):
```

```
    print(i) >>> 4 5 6 7 8 9
```

range(a, b, c) demek de a'dan b'ye kadar c'er c'er demektir:

```
for i in range(4,20, 5):
```

```
    print(i) >>> 4 9 14 19
```

range kullanarak liste olarak yazdırmak için de print(list(range()))

```
print(list(range(10,101,10))) >>> [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

range kullanarak liste olarak yazdırmanın bir diğer yolu:

```
numbers = [a for a in range(10)]
```

```
print(numbers)
```

```
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

bir başka yolu da append() metodudur:

```
numbers = [ ]
```

```
for a in range(10):
```

```
    numbers.append(a)
```

```
print(numbers)
```

```
>>> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

bir listenin elemanlarını istediğimiz işleme tabi tutarak yazdırma:

```
numbers = [a*2 for a in range(10)]
```

```
print(numbers)
```

```
>>>
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

normalde [a for a in range(10)] deseydik 1 ... 9 yazdıracaktı. fakat a*2 dediğimiz için her bir elemanı *2'ye tabi tutup yazdırdı.

bunların arasından bir ayırım yapmak istiyorsan:

```
numbers = [a**2 for a in range(10) if a % 3 == 0]
```

```
print(numbers)
```

```
>>> [0, 9, 36, 81] #bir koşul ekledik ve dedik ki range(10)dan yalnızca 3'e kalansız bölünenleri al.
```

örnek:

```
squares = [ ]
```

```
for value in range(1,11):
```

```
    square = value**2
```

```
    squares.append(square)
```

```
print(squares) >>> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

yukarıdakinin daha kısa hâli:

```
squares = [ ]
```

```
for value in range(1, 11):
```

```
    squares.append(value**2)
```

```
print(squares)
```

```
squares = [value**2 for value in range(1,11)]
print(squares) >>> [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

- ```
def func(a):
 return " ".join(i for i in a.split()[::-1])
print(func("bugun gunlerden pazar")) >>> pazar gunlerden bugun
burada kritik bilgi [::-1] olayıdır. bu tersten string almaya yarar.
```

- ```
def func(n):
    return int("".join(sorted(str(n), reverse=True)))
print(func(234009)) >>> 943200
```

- ```
liste1 = ["a" , 1, "alpha"]
liste2 = ["b" , 2, "b"]
listeSon = liste1 + liste2
print(listeSon) >>> ['a', 1, 'alpha', 'b', 2, 'b']
```

```
liste1 = ["a" , 10, "alpha"]
liste2 = ["b" , 23, "b"]
listeSon = [liste1 , liste2]
print(listeSon) >>> [['a', 10, 'alpha'], ['b', 23, 'b']]
```

```
a = [2, 3, 4, 1, 100, 24]
a.reverse()
print(a) >>> [24, 100, 1, 4, 3, 2]
```

- ```
def sesli_silici(string):
    return "".join([c for c in string if c not in "aeioöüü"])
print(sesli_silici("merhabalar"))
```

- ```
def greet(language):
 return {
 'czech': 'Vitejte',
 'danish': 'Velkomst',
 'dutch': 'Welkom',
 'turkish': 'hoş geldiniz',
 'english': 'Welcome'}.get(language, 'Welcome')
print(greet("turkish"))
```

```
def greet(language):
 greets = {
 'czech': 'Vitejte',
 'danish': 'Velkomst',
 'dutch': 'Welkom',
 'turkish': 'hoş geldiniz',
 'english': 'Welcome'
 }
 return greets[language] if language in greets else "welcome"
```

- `insert(a, b)` metodu `a`. indekse `b` elemanını eklemeni sağlar:

```
liste = [10, 20, 40, 50]
```

```
liste.insert(2, 30)
```

```
print(liste) >>> [10, 20, 30, 40, 50]
```

- `remove()` metodu takes a single element and helps you delete an element from a list:

```
liste = [10, 20, 40, 50]
```

```
liste.remove(20)
```

```
print(liste) >>> [10, 40, 50]
```

eğer sildiğin elemandan birden çok varsa listede sadece ilk elemanı siler.

- `extend()` method appends a whole list to another list:

```
a = ["ankara", "istanbul", "california"]
```

```
b = []
```

```
b.extend(a)
```

```
print(b) >>> ['ankara', 'istanbul', 'california']
```

- bir listenin elemanını değiştirmek için değiştirmek istediğin elemanın indeksiyle yeni bir değer atarsın:

```
liste = ["?", 2, 3, 4]
```

```
liste[0] = 1
```

```
print(liste) >>> [1, 2, 3, 4]
```

- bir listedeki duplicate elemanları kaldırmanın en kolay yolu:

```
list(set(liste))
```

fakat listeye çevirirken orijinal sıra bozulur. bozulmasın istiyorsan `orderedDict` kullanmalısın.

- bir listeden bir listeye koşullu eleman ekleme:

```
liste1 = [1, 2, 4, 5, 6, 7, 9, 12, 13]
```

```
liste2 = []
```

```
for a in liste1:
```

```
 if a % 2 == 1:
```

```
 liste2.append(a)
```

```
print(liste2) >>> [1, 5, 7, 9, 13]
```

yukarıdakinin aynısını tek satırda da yapabilirsin:

```
liste1 = [1, 2, 4, 5, 6, 7, 9, 12, 13]
```

```
liste2 = [a for a in liste1 if a % 2 == 1]
```

```
print(liste2)
```

- bu yöntemle `list comprehension` deniyor. şeması şöyle:

```
[expression for element in iterable if condition]
```

list comprehension örnek:

```
nums = [1,2,3,4]
```

```
squared_nums = [num * num for num in nums]
```

```
print(squared_nums) >>> [1, 4, 9, 16]
```

örnek2:

```
a = [i * 2 for i in (1,2,3,4)]
```

```
print(a) >>> [2, 4, 6, 8]
```

örnek3:

```
a = [s.upper() for s in "merhaba"]
```

```
print(a) >>> ['M', 'E', 'R', 'H', 'A', 'B', 'A']
```

örnek4:

```
a = [s.strip("z") for s in ["benimz", "adımz", "bugzra"]]
```

```
print(a)
```

görüldüğü gibi yalnızca sondakileri siler `strip()`

örnek5:

```
sentence = "the die is cast"
```

```
a = ["".join(sorted(word, key = lambda x: x.lower())) for word in sentence.split()]
```

```
print(a) >>> ['eht', 'dei', 'is', 'acst']
```

list comprehension'da else de kullanabilirsin ama `for`'dan önce kullanmalısın:

```
mesela bir stringdeki sessiz harfleri '*'a çevirelim
```

```
örn: apple >>> ['a', '*', '*', '*', 'e']
```

```
a = [x if x in 'aeiou' else '*' for x in 'bugrahan']
```

```
print(a) >>> ['*', 'u', '*', '*', 'a', '*', 'a', '*']
```

birden fazla if de kullanabilirsin:

```
a = [i for i in range(20) if i % 2 == 0 if i > 10]
```

```
print(a) >>> [12, 14, 16, 18]
```

iki list comprehension birleştirme:

```
a = [2 * (x if x % 2 == 0 else -1) + 1 for x in range(10)]
```

```
print(a) >>> [1, -1, 5, -1, 9, -1, 13, -1, 17, -1]
```

- append örneği:

```
a = ["kalem", "kitap", "cetvel"]
```

```
b = []
```

```
for i in a:
```

```
 if "a" in i:
```

```
 b.append(i)
```

```
print(b)
```

ya da list comprehensionla daha kısa yolu

```
a = ["kalem", "kitap", "cetvel"]
```

```
b = [i for i in a if "a" in i]
```

```
print(b) >>> ["kalem", "kitap"]
```

- list comprehension + range metodu:

```
newlist = [x for x in range(10)]
```

```
print(newlist) >>> 0 1 2 3 4 5 6 7 8 9
```

örnek2:

```
newlist = ["x" for x in range(5)]
```

```
print(newlist) >>> ['x', 'x', 'x', 'x', 'x']
```

örnek3:

```
oldlist = [1, 2, 3, 4, 5]
```

```
newlist = ["merhaba" for x in oldlist]
```

```
print(newlist) >>> ['merhaba', 'merhaba', 'merhaba', 'merhaba', 'merhaba']
```

örnek4:

```
oldlist = [1, 2, 3, 4, 5]
```

```
newlist = [x if x % 2 == 0 else x % 2 != 1 for x in oldlist]
```

```
print(newlist) >>> [False, 2, False, 4, False]
```

- bir listeye yeni eleman eklemek:

```
araba = ["bmw", "mercedes", "opel", "mazda"]
```

```
result = araba + ["audi", "nissan"]
```

```
print(result) >>> ['bmw', 'mercedes', 'opel', 'mazda', 'audi', 'nissan']
```

- del metoduyla listeden indeks numarası ile eleman sileriz:

```
araba = ["bmw", "mercedes", "opel", "mazda"]
```

```
del araba[1]
```

```
result = araba
```

```
print(result) >>> ['bmw', 'opel', 'mazda']
```

- min() ve max() metodları bir listedeki sayı olarak en küçük ya da alfabetik olarak ilk geleni verir:

```
a = [134, 42, 324, 544]
```

```
print(min(a)) >>> 42
```

ya da

```
b = min(a)
```

```
print(b) >>> 42
```

- kaç yılında 100 yaşında olacağını hesaplayan program:

```
import datetime
```

```
a = datetime.datetime(2021, 2, 2)
```

```

year = a.strftime("%Y")
name = input("What's your name?:")
age = int(input("How old are you?: "))
birthyear = int(year) - age
print(f"Hi {name}. You'll be 100 in {birthyear+100}")

```

- x in y metodu bir listede bir kelimenin/harfin/sayının olup olmadığını sorgulamamızı sağlar:

```

araba = ["bmw" , "mercedes" , "opel" , "mazda"]
result = "mercedes" in araba
print(result) >>> True

```

- liste içinde liste yazmaya nested list denir:

```

nums = [[1, 2], [3, 4], [5, 6]]
print(nums[2][0]) >>> 5

```

you can enrich this method:

```

nums = [[1, 2, [3, 4, [5, 6]]]]

```

- in Python, we are allowed to extract the values back into variables. This is called "unpacking":

```

meyve = ["elma", "armut", "kiraz"]
(sarı, kırmızı, mavi) = meyve
print(sarı)
print(kırmızı)
print(mavi) >>>
elma
armut
kiraz

```

Note: The number of variables must match the number of values in the tuple, if not, you must use an asterix to collect the remaining values as a list. If the number of variables is less than the number of values, you can add an \* to the variable name and the extra values will be assigned to the variable as a list:

```

meyve = ["elma", "armut", "kiraz", "üzüm", "çilek"]
(sarı, kırmızı, *mavi) = meyve # sırayla yazdığımız bu key'ler meyve listesinin
elemanlarına sırayla atandı. fakat liste elemanları key adedinden çok olduğu için elemanın
başına * koyarak kalanını liste olarak yazdırdık.
print(sarı)
print(kırmızı)
print(mavi) >>>
elma
armut
['kiraz', 'üzüm', 'çilek']

```

If the asterix is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left:

```

meyve = ["elma", "armut", "kiraz", "üzüm", "çilek"]
(*sarı, kırmızı, mavi) = meyve
print(sarı)
print(kırmızı)
print(mavi) >>>
['elma', 'armut', 'kiraz']
üzüm
çilek

```

- enumerate() metodu ile listenin elemanlarını indeks numaralarıyla yazdırma:

```

list = ["john","beowulf","clara","emily","susan","gervais"]
for index, element in enumerate(list):
 print("{ } has an index of { }".format(element, index)) >>>
john has an index of 0

```

```
beowulf has an index of 1
clara has an index of 2
emily has an index of 3
susan has an index of 4
gervais has an index of 5
```

- 0 yazana kadar girilen sayıların ortalamasını bulma:

```
sayilar = []
while True:
 sayi = int(input("bir sayi giriniz:"))
 if sayi == 0:
 break
 sayilar.append(sayi)
toplam = sum(sayilar)
print("sayilarin toplami =", toplam / len(sayilar))
```

- bir listenin elemanlarının türünü değiştirme:

```
items = ["1","2","3","4"]
a = [int(item) for item in items] >>>
```

- In Python, we can implement a matrix as nested list (list inside a list). We can treat each element as a row of the matrix. For example X = [[1, 2], [4, 5], [3, 6]] would represent a 3x2 matrix.

- iki listeyi birleştirme:

```
liste1 = [1, 2, 3, 4, 5]
liste2 = ["a","b", "c"]
liste3 = liste1 + liste2
print(liste3) >>> [1, 2, 3, 4, 5, 'a', 'b', 'c']
```

- append() metoduyla listeye eleman ekleme:

```
liste1 = [1, 2, 3, 4, 5]
liste2 = ["a","b", "c"]
for a in liste2:
 liste1.append(a)
print(liste1) >>> [1, 2, 3, 4, 5, 'a', 'b', 'c']
```

örnek:

```
liste = ["bugra", "kara", 123, 250, "cüzdan"]
liste.append("siyah")
print(liste) >>> ['bugra', 'kara', 123, 250, 'cüzdan', 'siyah']
```

loop + append() kullanımı:

```
cubes = []
for i in range(1,11):
 cube = i ** 3
 cubes.append(cube)
print(cubes) >>> [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

aynısının daha kolay yolu:

```
cubes = [number ** 3 for number in range(1,11)]
print(cubes)
```

- index() metoduyla bir listedeki bir elemanın indexini buluruz.

```
sehirler = ["samsun", "ankara", "istanbul", "manisa"]
a = sehirler.index("manisa")
print(a) >>> 3
```

- print ile yazdırma örnekleri:

```
print("29", "01", "2022", sep="/") >>> 29/01/2022
```

- You can separate big numbers with the underscore This way it can be easier to read them:

```
print(1_000_000_000) >>> 1000000000
```

- sort() metodu alfabetik ya da numerik olarak listenin elemanlarını sıralar.



```
liste = ["bugra", "kara", "cüzdan"]
liste.sort()
print(liste) >>> ['bugra', 'cüzdan', 'kara']
ya da sayısal olarak:
liste = [23423, 7675, 123, 100, -323, -15245]
liste.sort()
print(liste) >>> [-15245, -323, 100, 123, 7675, 23423]
```

sort() method is case sensitive, resulting in all capital letters being sorted before lower case:

```
list = ["ahmet", "Zilan", "ceylin", "hatice"]
list.sort()
print(list) >>> ['Zilan', 'ahmet', 'ceylin', 'hatice']
```

- sort(reverse = True) metodu bir listenin elemanlarını tersten sıralar:

```
list = ["ahmet", "zilan", "ceylin", "hatice"]
list.sort(reverse = True)
print(list) >>> ['zilan', 'hatice', 'ceylin', 'ahmet']
```

- reverse() metodu alfabetik/numerik bakmaksızın verilen liste sırasını tersine çevirir:

```
list = [1, 2, 0, 3, 4, 2, 44, 20]
list.reverse()
print(list) >>> [20, 44, 2, 4, 3, 0, 2, 1]
```

sort()’un tersi değildir. sort()’un tersi sort(reverse = True)’dur. reverse() yazdığın sırayı tersine çevirir.

- to check if a list is empty:

```
my_list = []
if not my_list:
 print("the list is empty")
```

- insert() metodu da append() metodu gibi çalışır ama bunda istediğin indeksten öncesine istediğin şeyi ekleyebiliyorsun:

```
liste = ["bugra", "kara", 123, 250, "cüzdan"]
liste.insert(4, "siyah bir")
print(liste) >>> ['bugra', 'kara', 123, 250, 'siyah bir', 'cüzdan']
```

listenin sonuna eklemek istiyorsan a.insert(-1, x) demelisin. çünkü bir listenin son elemanının indeksi -1’dir.

- extend() metoduyla bir listeden bir başka listeye eleman ekleyebilirsiniz:

```
liste1 = ["volvo", "hyundai", "bmw"]
liste2 = ["harley", "yamaha"]
liste1.extend(liste2)
print(liste1) >>> ['volvo', 'hyundai', 'bmw', 'harley', 'yamaha']
```

- pop() metoduyla bir listenin sonundaki elemanı silebilirsin:

```
liste = ["bugra", "kara", 123, 250, "cüzdan"]
liste.pop()
print(liste) >>> ['bugra', 'kara', 123, 250]
```

kaçıncı index’teki elemanı silmek istiyorsan onun index’ini pop()’un içine yazabilirsin. örnek:

```
langs = ["Python", "Ruby", "Perl", "Lua", "JavaScript"]
print(langs)
deleted = langs.pop(3)
print("{0} was removed".format(deleted)) >>>
['Python', 'Ruby', 'Perl', 'Lua', 'JavaScript']
Lua was removed
```

- clear() metodu bir listenin içeriğini boşaltır:

```
n = [3, 4, 1, 7, 2, 5, 8, 6]
n.clear()
print(n) >>> []
```

- list + loop kullanımı:

```
guests = ["Martin Luther" , "Kobe Bryant", "Doctor Who"]
for i in guests:
 print("Hoş Geldiniz" , i)
>>>
Hoş Geldiniz Martin Luther
Hoş Geldiniz Kobe Bryant
Hoş Geldiniz Doctor Who
```

- double print usage for loops:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
 print(magician.title() + ", that was a great trick!")
 print("I can't wait to see your next trick, " + magician.title() + ".") >>>
Alice, that was a great trick!
I can't wait to see your next trick, Alice.
David, that was a great trick!
I can't wait to see your next trick, David.
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

if you want to print something after the loop is finished, write it unindented, other wise it will be included in the loop block.

- sum() metoduyla bir listedeki sayıları toplama:

```
numbers = [1,2,3,4,5,6,7,8,9]
print(sum(numbers)) >>> 45
```

- string ifadeyi listeye çevirme:

```
string = "bugra,kara,19"
liste = string.split(",") #virgülden ayrılırsın istiyorsan virgül yazarsın.
print(liste) >>> ['bugra', 'kara', '19']
```

- there are 8 methods to copy a list in python:

- o copy() metodu
- o [:] metodu
- o list() metodu
- o = metodu (atama)
- o extend() metodu
- o deepcopy() metodu
- o for + append() metodu
- o list comprehension metodu

1.

```
liste_a = ["isim","şehir","hayvan","bitki"]
liste_b = liste_a.copy()
```

2.

```
liste_a = ["isim","şehir","hayvan","bitki"]
liste_b = liste_a[:]
```

3.

```
liste_a = ["isim","şehir","hayvan","bitki"]
liste_b = list(liste_a)
```

4.

```
liste_a = ["isim","şehir","hayvan","bitki"]
liste_b = liste_a
```

5.

```
n = ["isim","şehir","hayvan","bitki"]
m = []
m.extend(n)
print(m)
```

6.

```
import copy
n = ["isim", "şehir", "hayvan", "bitki"]
m = copy.deepcopy(n)
print(m)
```

7.

```
n = ["isim", "şehir", "hayvan", "bitki"]
m = []
for i in n:
 m.append(i)
```

8.

```
n = ["isim", "şehir", "hayvan", "bitki"]
m = [i for i in n]
```

- bir listeyi tersten yazdırmak için `[::-1]` kullanırız:

```
a = [1,2,3,4,5,6,7,8]
print(a[::-1]) >>> [8, 7, 6, 5, 4, 3, 2, 1]
```

- bir listenin içindeki duplicate olanları kaldırmak için `fromkeys` ile önce dictionary'e sonra tekrar list'e çeviririz:

```
mylist = ["a", "b", "a", "c", "c"]
mylist = list(dict.fromkeys(mylist))
print(mylist) >>> ['a', 'b', 'c']
```

sözlüğe çevrilince mükerrer elemanlar silinir ve o hâlini listeye çeviririz.

- kullanıcıdan alınan bilgiyle liste oluşturma:

```
markaListesi = []
marka1 = input("marka1: ")
marka2 = input("marka2: ")
markaListesi.append(marka1)
markaListesi.append(marka2)
print(markaListesi) >>> ['opel', 'toyota']
```

`append` metodu yalnızca bir argument alır, yani `.append(marka1, marka2)` diyemiyoruz. alt alta yazmak zorundayız.

- bir listeden ilk n kadar elemanı yazdıran program:

```
def take(arr,n):
 return arr[:n]
```

- eğer sıralı dictionary istiyorsan `OrderedDict` modülü kullanmalısın:

```
from collections import OrderedDict
d = OrderedDict()
d['first'] = 1
d['second'] = 2
d['third'] = 3
d['last'] = 4
Outputs "first 1", "second 2", "third 3", "last 4"
for key in d:
 print(key, d[key])
```

- bir listenin boş olup olmadığını anlamak için `if len(list) == 0` demeye gerek yoktur:

```
list = []
if not list:
 print("this is empty")
```

çünkü boş bir listenin değeri `False` yani `not`'tır zaten. dolu olup olmadığını anlamak için de:

```
list = []
if list:
 print("this is not empty")
```

'if list' means if list is NOT empty

- all() metodu bir listenin bütün elemanları True ise True verir, biri bile False ise False verir:

```
list = [True, 1, 1, "a"]
print(all(list)) >>> True
```

fakat herhangi bir eleman 0, "" , None ya da False olsaydı False verirdi

any() metodu bir listenin içinde en az bir tane True ve muadili değer varsa True verir:

```
list = [None, None, 0, False, 0, 1]
print(any(list)) >>> True
```

any() örnek:

```
list = [3, 4, 5, 6]
print(any(val % 3 == 0 for val in list))
```

-

```
list = [30, 40, 50, 60]
print(any(val % 9 == 0 for val in list))
```

- zip() metodu returns a list of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables:

```
alist = ['a1', 'a2', 'a3']
blist = ['b1', 'b2', 'b3']
for a, b in zip(alist, blist):
 print(a, b) >>>
a1 b1
a2 b2
a3 b3
```

#tuple

- tuple methods:
  - o count()
  - o index()
- Tuples are used to store multiple items in a single variable.
- A tuple is a collection which is ordered and unchangeable (also immutable).
- Tuples are written with round brackets.
- Since tuples are indexed, they can have items with the same value:

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

- tek elemanlı öğelerin str değil de tuple olması için elemandan sonra virgül getirmelisin:

```
a = ("bugra",)
print(type(a)) >>> tuple. virgül olmazsa str algılıyor
```

- listelerde yaptığımız
  - list[0] = "farklı bir eleman"
 atamasını tuple'lerde yapamıyoruz. onun dışında listelerle aynı.
- tuple'lerde parantez kullanmak zorunda değilsin:

```
tupleOrnek = "bugra" , 12, True
```

- The tuple can contain mixed data types, including tuples (then it is called nested tuples):

```
tuple = ("bugra", "kara", (1,2,3,4))
print(tuple[2][3]) >>> 4
```

- what does it mean "ordered" ? : When we say that tuples are ordered, it means that the items have a defined order, and that order will not change. since tuples are unchangeable, if you want to change or add sth to a tuple, you may first turn it into a list, change it by index[ ] = method and then change it into a tuple again:

```
a = ("bugra", 13, "yasinda")
b = list(a)
b[1] = 19
a = tuple(b)
print(a) >>> ('bugra', 19, 'yasinda')
```

- örnek:

```

meyve = ("bugra", "kara", 19)
for i in range(len(meyve)):
 print(meyve[1]) >>> #if you say meyv[i] it will print all elemans
kara
kara
kara

```

- tuple'ları birleştirmek için (listeler gibi) tuple3 = tuple1 + tuple2 diyebilirsiniz.
- bir listedeki elemanların sadece ilk harflerini yazdıran program:

```

list = ["john", "beowulf", "clara", "emily", "susan", "gervais"]
for s in list:
 print(s[:1])

```

## #dictionary

- A Python dictionary is a group of key-value pairs. The elements in a dictionary are indexed by keys. Keys in a dictionary are required to be unique. that is why two values cannot have same key. dictionary'ler, key : value mantığıyla çalışır. yani bir bilgi [key] (i.e brand) dictionary'de bir başka bilgiyi [value] ("Ford") temsil eder. curly brackets içinde yazılır, key ile value arasına = değil : konur. keyler virgül ile ayrılır. şeması şu şekildedir:

```

thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964 }

```

- dictionary methods:
  - o clear()
  - o copy()
  - o fromkeys()
  - o get()
  - o items()
  - o keys()
  - o popitem()
  - o setdefault()
  - o pop()
  - o values()
  - o update()
- As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered. When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.
- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- value kısmı bütün veri türünü içerebilir fakat keys kısmı sadece string ve int tipinde olabilir.
- beş farklı dictionary oluşturma yolu:

1. normal { } içinde key ve value atayarak

```

o yazarlar = {"tolstoy": "Rus", "balzac": "Fransız", "Dickens": "İngiliz"}

```

2. dict() metoduyla key'ler keyword olacak şekilde

```

o yazarlar = dict(tolstoy="rus", balzac="fransız", dickens="ingiliz")

```

3. boş bir sözlük yaratıp daha sonra [] ile ekleyerek

```

o yazarlar = { }
yazarlar["tolstoy"] = "rus"
yazarlar["balzac"] = "fransız"
yazarlar["dickens"] = "ingiliz"

```

eğer bir value liste ise (possible) dict1["key"].append(value) şeklinde yeni bir değer ekleyebilirsiniz o listeye

#### 4. dictionary comprehension kullanarak

```
sayilar = {i: object() for i in range(4)}
```

#### 5. fromkeys() metodu ile

```
dictionary = dict.fromkeys((milk, eggs), (2, 5)) >>> {'milk': 2, 'eggs': 5}
```

- dictionary comprehension:

```
d = {k:v for k, v in [('a', 1)]}
print(d) >>> {'a': 1}
```

örnek:

```
a = {x: x*x for x in (1,2,3,4)}
print(a) >>> {1: 1, 2: 4, 3: 9, 4: 16}
```

- dictionary'e yeni bir eleman eklemek için you say simply:

```
dict["newKey"] = newValue (can be a list, int, dict etc.)
```

- Duplicate keys will overwrite existing key:

```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964,
 "year": 2020
}
print(thisdict) >>> {'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

- Unlike lists, items in dictionaries are unordered:

```
a = {"isim": "bugra", "yaş": 21, "il": "istanbul"}
b = {"yaş": 21, "il": "istanbul", "isim": "bugra"}
print(a == b) >>> True
```

- bool() metoduyla bir dictionary'nin empty olup olmadığını sorgulayabilirsiniz:

```
bosSozluk = { }
a = bool(bosSozluk)
print(a) >>> False (boş olduğu için, dolu olsa True verirdi)
```

- eğer belirli bir key'in value'sini çağırmak istiyorsan:

```
print(dict1 ["key_ismi"])
aynı şeyi get() metoduyla da yapabilirsin
```

```
x = dict1.get("key_ismi")
print(x)
```

get() metodu örnek:

```
plakalar = {"samsun": 55, "izmir": 35, "manisa": 15, "istanbul": 34}
print(plakalar.get("samsun")) >>> 55
```

eğer get metodunu iki parametrelili yazarsan ve yazdığın ilk parametre key olarak dict'de mevcut değilse 2. parametreyi yazdırır ekrana:

```
plakalar = {"samsun": 55, "izmir": 35, "manisa": 15, "istanbul": 34}
print(plakalar.get("adana", "bulunamadı")) >>> bulunamadı
```

hata vermesini önlemenin 2. bir yolu da try:except kullanmaktır:

```
try:
 value = dict[key]
except:
 value = defaultValue
```

- { }.fromkeys() method creates a new dictionary from a list/tuple using the same value entered as the 2<sup>nd</sup> argument:

```
basket = ['oranges', 'pears', 'apples', 'bananas']
fruits = { }.fromkeys(basket, "meyve")
print(fruits) >>> {'oranges': 'meyve', 'pears': 'meyve', 'apples': 'meyve', 'bananas': 'meyve'}
```

örnek:

```
sayi = (10, 14, 64)
sayi_dict = { }.fromkeys(sayi, "is a çift sayı")
print(sayi_dict)
```

- setdefault() method returns a value if the key you entered as first parameter exists in the dict. If it does not, it creates a new element in the dictionary with the value that you entered as the 2<sup>nd</sup> parameter and prints this value.

```
plakalar = {"samsun": 55, "izmir": 35, "manisa": 15, "istanbul": 34}
print(plakalar.setdefault("zonguldak", 49))
print(plakalar) >>>
49
{'samsun': 55, 'izmir': 35, 'manisa': 15, 'istanbul': 34, 'zonguldak': 49}
```

yani fromkeys() metodu sadece stringlerden oluşan bir tuple'dan 2. parametre olarak girdiğin değerin bu stringlere value olduğu bir dictionary yaratmak için kullanılır.

- The chess pieces are identified by letters: K for king, Q for queen, R for rook, B for bishop, and N for knight. Describing a move uses the letter of the piece and the coordinates of its destination. A pair of these moves describes what happens in a single turn (with white going first); for instance, the notation 2. Nf3 Nc6 indicates that white moved a knight to f3 and black moved a knight to c6 on the second turn of the game.
- Üç metod ile bir dictionary'nin elemanlarına erişebilirsiniz:
  - o keys() metodu bütün key'leri liste halinde verir.
  - o values() metodu da value'ları liste halinde verir.
  - o items() metodu da hem key hem value'ları dictionary halinde verir.

örnek:

```
words = { 'girl': 'Maedchen', 'house': 'Haus', 'death': 'Tod' }
print(words.keys()) >>> dict_keys(['girl', 'house', 'death'])
print(words.values()) >>> dict_values(['Mädchen', 'Haus', 'Tod'])
print(words.items()) >>> dict_items([('girl', 'Mädchen'), ('house', 'Haus'), ('death', 'Tod')])
```

- dictionary'lerde in metodunu kullanabilirsin fakat in metodu yalnızca key olarak arar, value sorarsan olumsuz yanıt verecektir:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
if "bugra" in dictionary:
 print("evet, soyisim key'i dictionary sözlüğünün içinde")
else:
 print("hayır") >>> hayır
```

- update() metoduyla dictionary key ya da value'larını değiştirilebilirsin:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
dictionary.update({"yaş": 23})
print(dictionary) >>> {'isim': 'bugra', 'soyisim': 'kara', 'yaş': 23}
```

- pop("keyName") metodu ile girdiğin key ve onun value'su silinir. :

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
dictionary.pop("isim")
print(dictionary) >>> {'soyisim': 'kara', 'yaş': 19}
```

- popitem() metodu son girilen key.value'yu siler:

```
a = {"ad" : "bugra", "soyad" : "kara", "yaş" : 19, "şehir" : "istanbul" , "okul" : "Üni"}
a.popitem()
print(a) >>> {'ad': 'bugra', 'soyad': 'kara', 'yaş': 10, 'şehir': 'istanbul'}
```

- del() metoduyla da silebilirsin:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
del dictionary["yaş"]
print(dictionary) >>> {'isim': 'bugra', 'soyisim': 'kara'}
```

ya da del dictionaryName ile komple de silebilirsin.

- dictionary'de loop'lar:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
for i in dictionary:
 print(i) >>>
isim
soyisim
yaş
```

dikkat edilirse yalnızca key'leri yazdırdı. bunu comprehension ile yapmanın yolu:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
print([key for key in dictionary])
```

value'ları yazdırmak için i'yi [ ] içinde yazdırmalısın:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
for i in dictionary:
 print(dictionary[i]) >>>
bugra
kara
19
```

ya da values() metoduyla da value'leri yazdırabilirsin:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
for i in dictionary.values():
 print(i) >>>
bugra
kara
19
```

aynısını dictionary.keys() ile key'ler için de yapabilirsin.

```
dictionary = {"toyota": "araba", "boeing": "uçak", "harley": "motor"}
for i in dictionary.keys():
 print(i) >>>
toyota
boeing
harley
```

her ikisini de yazdırmak içinse items() metodunu kullanmalısın:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
for i in dictionary.items():
 print(i) >>>
('isim', 'bugra')
('soyisim', 'kara')
('yaş', 19)
```

örnek:

```
yemek = {
 "elma" : "amasya",
 "kırPidesi" : "samsun",
 "kebab" : "adana",
 "köfte" : "inegöl"
}
for a in yemek:
 print(f' {a} is famous in {yemek[a]}') >>> #value yazdırmak için yemek[a] key
yazdırmak için a yazıyoruz
elma is famous in amasya
kırPidesi is famous in samsun
kebab is famous in adana
köfte is famous in inegöl
```

o fakat burada dictionary olarak çağıramazsın, o yüzden for a in meyve(): yerine meyve: demek zorundasın.

örnek2:



```
favourite_languages = {
 "bugra" : "python",
 'ahmet' : "ruby",
 "mehmet" : "solidity",
 'ayşe' : 'C++' }
for i in favourite_languages:
 print(f"{i}'s favorite language is {favourite_languages[i]}") >>>
bugra's favorite language is python
ahmet's favorite language is ruby
mehmet's favorite language is solidity
ayşe's favorite language is C++
```

örnek3:

```
dictionary = {
 "nick" : "eskoda",
 "password" : "bugra2021",
 "age" : "19" }
for key, value in dictionary.items():
 print("Key: " + key)
 print("Value: " + value) >>>
Key: nick
Value: eskoda
Key: password
Value: bugra0855
Key: age
Value: 19
```

o key ve value yerine a, b de diyebilirsiniz. keyword değildirler.

örnek4:

```
l = {
 "turkey":"ankara",
 "england":"london",
 "usa":"washington dc",
 "netherland":"amsterdam"
}
for k in l.keys():
 print("{ } is the capital of { }.".format(l[k], k))
```

- how to swap (ters çevirmek) keys and values of a dictionary:

```
dict = {'a': 1, 'b': 2, 'c': 3}
reversed_dict = {j: i for i, j in dict.items()}
print(reversed_dict) >>> {1: 'a', 2: 'b', 3: 'c'}
```

- you can and sometimes should do nesting in dictionaries with lists:

```
kişi1 = {"ad" : "bugra" , "yaş" : 19 , "şehir" : "samsun"}
kişi2 = {"ad" : "cemre" , "yaş" : 25 , "şehir" : "adana"}
kişi3 = {"ad" : "ahmet" , "yaş" : 50 , "şehir" : "newşehir"}
kişilistesi = [kişi1, kişi2, kişi3]
for a in kişilistesi:
 print(a) >>>
{'ad': 'bugra', 'yaş': 19, 'şehir': 'samsun'}
{'ad': 'cemre', 'yaş': 25, 'şehir': 'adana'}
{'ad': 'ahmet', 'yaş': 50, 'şehir': 'newşehir'}
```

- for programming languages, everything is numbers. Even letters and symbols. They are just special types stored in special tables, such as ASCII. her şey; sayılar, renkler, harfler, semboller en temelde 0 ve 1'den oluşur.
- ord() metoduyla bir şeyin ASCII değerini bulabilirsiniz:

```
print(ord("a")) >>> 97
```

- iterate a dictionary:

```
dt = {'a': 'juice', 'b': 'grill', 'c': 'corn'}

for key, value in dt.items():
 print(key, value)

for key in dt:
 print(key, dt[key])
```

- A dictionary can contain dictionaries, this is called nested dictionaries:

```
myfamily = {
 "child1" : {
 "name" : "Emil",
 "year" : 2004
 },
 "child2" : {
 "name" : "Tobias",
 "year" : 2007
 },
 "child3" : {
 "name" : "Linus",
 "year" : 2011
 }
}
```

or, if you want to add three dictionaries into a new dictionary, create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {
 "name" : "Emil",
 "year" : 2004
}
child2 = {
 "name" : "Tobias",
 "year" : 2007
}
child3 = {
 "name" : "Linus",
 "year" : 2011
}
myfamily = {
 "child1" : child1,
 "child2" : child2,
 "child3" : child3
}
```

- dictionary'lerde loop kullanımı:

```
pizza = {
 "kenar" : "ince",
 "ekstra" : ["mantar", "sisis", "sucuk", "salam"]
}

print(f'sipariş verdiginiz pizza {pizza["kenar"]} kenarlı ve ekstra malzemeleriniz: ')
for i in pizza["ekstra"]:
 print(i) >>>
sipariş verdiginiz pizza ince kenarlı ve ekstra malzemeleriniz:
mantar
sisis
sucuk
```

```
salam
```

örnek:

```
languages = {
 "bugra" : ["c2 turkish", "c1-2 english", "a1 german"],
 "ahmet" : ["c2 turkish", "b1 english", "b2 russian",],
 "albrecht" : ["c2 german", "a1 english"],
}
for a, b in languages.items():
 print(f"{a}'nın bildiği diller: ")
 for c in b:
 print(c)
```

örnek2:

```
sevdiği_film_ahmet = {
 "ahmet" : ["the godfather", "z report", "world at war"],
 "mehmet" : ["les miserables" , "third man"]
}
for a, b in sevdiği_film_ahmet.items():
 print(f"{a}'nın en sevdiği filmler: ")
 for c in b:
 print(f"\t {c}'dir.") >>>
ahmet'nın en sevdiği filmler:
 the godfather'dir.
 z report'dir.
 world at war'dir.
mehmet'in en sevdiği filmler:
 les miserables'dir.
 third man'dir.
```

- dictionary veri tipi bir ifadede değer atama, sonra key'ini girerek değeri yazdırma:

```
plakalar = {"samsun" : 55, "edirne" : 32, "karaman" : 13}
print(plakalar["karaman"]) >>> 13
```

- dictionary'ye yeni bir eleman eklemek için:

```
plakalar = {"samsun" : 55, "edirne" : 32, "karaman" : 13}
plakalar["ankara"] = 6
print(plakalar) >>> {'samsun': 55, 'edirne': 32, 'karaman': 13, 'ankara': 6}
varolan elemanı değiştirmek için de aynısı, üstüne otomatik yazdıracaktır.
```

örnek:

```
dictionary = {"isim" : "bugra", "soyisim": "kara" , "yaş" : 19}
dictionary["yaş"] = 25
print(dictionary) >>> {'isim': 'bugra', 'soyisim': 'kara', 'yaş': 25}
```

- input metodu ile dictionary doldurma:

```
ogrenciler = { }
number = input("no giriniz: ")
name = input("adi giriniz: ")
surname = input("soyadi giriniz: ")
telefon = input("telefon giriniz: ")
ogrenciler[number] = {
 "ad" : name,
 "soyad" : surname,
 "telefon" : telefon
}
print(ogrenciler) >>> {'125': {'ad': 'bugra', 'soyad': 'kara', 'telefon': 5058415239}}
```

- çoğu zaman ilk etapta boş bir liste tanımlayıp daha sonra ona key&value eklemek daha mantıklıdır.
- | metodu ile iki dictionary birleştirme:

```
dict1 = {"isim": "bugra", "yaş": 19}
dict2 = {"konum": "artvin", "mood": "moody"}
result = dict1 | dict2
print(result) >>> {'isim': 'bugra', 'yaş': 19, 'konum': 'artvin', 'mood': 'moody'}
```

**\*\*** metodu ile iki dictionary birleştirme:

```
dict1 = {"isim": "bugra", "yaş": 19}
dict2 = {"konum": "artvin", "mood": "moody"}
merged = {**dict1, **dict2}
print(merged) >>> {'isim': 'bugra', 'yaş': 19, 'konum': 'artvin', 'mood': 'moody'}
```

- defaultdict() metodu is used when keys for which no value has been explicitly defined can be accessed without errors. mesela aşağıdaki gibi bir sözlük olsun:

```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964 }
```

bu sözlükte varolmayan bir key'e erişmeye çalışırsan hata verecektir:

```
print(thisdict["kilometre"])
```

fakat defaultdict kullanırsan..:

```
from collections import defaultdict
```

```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964 }
```

```
thisdict = defaultdict(lambda: "bilinmiyor") #1
```

```
print(thisdict['kilometre']) #2
```

- o 1'de varolmayan keyler için default bir değer atadık
  - o 2'de varolmayan bir keyi yazdırmaya çalıştığımızda o değeri verdi
- {\*\*dict} metodu iki dictionary'yi birleştirir:

```
dict1 = {"name": "clair", "surname": "karloff", "age": 29}
dict2 = {"place": "UK", "job": "doctor", "surname": "johnson", "age": 36}
dict3 = {**dict1, **dict2}
print(dict3) >>> {'name': 'clair', 'surname': 'johnson', 'age': 36, 'place': 'UK', 'job': 'doctor'}
```

- o gördüğün gibi aynı key'leri yazarsan sonradan yazdığın bir öncekini override eder
-

# muhtelif

- renkli yazı yazdıran program

```
from termcolor import colored

print(colored('bugra', 'yellow'))
```

- string to dateTime converter

```
from dateutil import parser

date_time = parser.parse("Mar 11 2011 11:31AM")

print(date_time)
```

- Randomly Select an Element From the List

```
import random

my_list = [1, 'a', 32, 'c', 'd', 24]
print(random.choice(my_list))

or if you want cryptographically a safer method:
import secrets

my_list = [1, 'a', 32, 'c', 'd', 24]
print(secrets.choice(my_list))
```

- Convert Two Lists Into a Dictionary

```
index = [1, 2, 3]
languages = ['python', 'c', 'c++']

dictionary = dict(zip(index, languages))
print(dictionary)

or you can use list comprehension:
index = [1, 2, 3]
languages = ['python', 'c', 'c++']

dictionary = {k: v for k, v in zip(index, languages)}
print(dictionary)
```

- countdown:

```
import time

def countdown(time_sec):
 while time_sec:
 mins, secs = divmod(time_sec, 60)
 timeformat = '{:02d}:{:02d}'.format(mins, secs)
 print(timeformat, end='\r')
 time.sleep(1)
 time_sec -= 1
 print("stop")
countdown(5)
```

- while örnek:

```
is_on = True

while is_on:
 choice = input('What would you like to say: ')
 if choice == 'exit'.lower():
 is_on = False
```

bu kod exit yazana kadar yukarıdaki soruyu sormaya devam eder

## #sets

- set methods:
  - clear()
  - difference()
  - discard()
  - intersection()
  - isdisjoint() → ortak elemanları var mı
  - issubset()
  - issuperset()
  - pop()
  - symmetric\_difference()
  - union()
  - update()
- Sets are used to store multiple items in a single variable. A set is a collection which is both unordered and unindexed. Sets are written with curly brackets ama dictionary'ler gibi peşine : ile bir "value" almazlar. yani A set is a collection of elements with no repeats and without insertion order but sorted order. They are used in situations where it is only important that some things are grouped together, and not what order they were included. For large groups of data, it is much faster to check whether or not an element is in a set than it is to do the same for a list.

- özellikleri:

- süslü parantezle yazılır.

```
a = {"bugra","kara","molla"}
```
- unordered ve unindexed'dirler
- elemanları değiştirilemezler
- add() ve remove() metoduyla eleman eklenip çıkarılabilir

- bir set'in elemanları unordered olduğu için her yazdığında rastgele sırayla yazdırır.
- sets do not allow duplicate values, if you write in them, they only print single from everything:

```
set = {10, 10, 10, 10, 10, 10}
print(set) >>> {10}
```

- Once a set is created, you cannot change its items, but you can add new items.
- sets may be of any data types:

```
set = {25, True, "merhaba", 10.5}
print(set)
```

- örnek:

```
set1 = {'a', 'b', 'c', 'c', 'd'}
set2 = {'a', 'b', 'x', 'y', 'z'}
print("intersection: ", set1 & set2)
print("union: ", set1 | set2)
print("difference: ", set1 - set2)
print("symmetric difference: ", set1 ^ set2)
```

- The intersection operation returns elements that are both in set1 and set2.
- The union operation returns all elements from both sets.
- The difference returns elements that are in the set1 but not is set2.
- the symmetric difference returns elements that are in set1 or set2, but not in both.

- yani

- List is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered and unindexed. No duplicate members.
- Dictionary is a collection which is ordered\* and changeable. No duplicate members.

- örnek:

```
thisSet = {"apple", "banana", "cherry"}
print("pineapple" in thisset) >>> False
```

- elemanlarına alt alta döngü (for) ile ulaşılabilir:

```
a = { "elma" , "armut", "cilek"}
for i in a:
 print(i)
```

- add() metoduyla eleman eklenebilir:

```
a = { "elma" , "armut", "cilek"}
a.add("kiraz")
print(a) >>> {'cilek', 'elma', 'armut', 'kiraz'}
```

- update() method is used to add items from another set into the current set. The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

```
a = { "elma" , "armut", "cilek"}
a.update(["kiraz","üzüm"])
print(a) >>> {'armut', 'cilek', 'elma', 'kiraz', 'üzüm'}
```

direkt bir başka seti de ekleyebilirsiniz:

```
set1 = {"merhaba", "şişe", "kalem", 123}
set2 = {"kamera", "kuş", False}
set1.update(set2)
print(set1) >>> {'kalem', False, 'kuş', 'kamera', 'şişe', 'merhaba', 123}
```

- remove() metoduyla içinden eleman silebilirsiniz.

```
a = [1, 2, 2, 33, 33, 250, 250, 100]
a.remove(100)
a = set(a)
print(a) >>> {1, 2, 33, 250}
```

- discard() metoduyla set'lerden eleman silebilirsiniz:

```
a = { "elma" , "armut", "cilek"}
a.discard("armut")
print(a) >>> {'cilek', 'elma'}
```

remove() - discard() farkı, eğer sileceğin eleman listede yoksa remove hata verir, discard vermez.

- clear() metodu kümeyi temizlemeye yarar:

```
kume ={1,2,3,4,5,6}
kume.clear()
print(kume)>>> set()
```

- You can use the union() method that returns a new set containing all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3) >>> {1, 'a', 2, 3, 'b', 'c'}
```

or the update() method that inserts all the items from one set into another:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1) >>> {'a', 1, 2, 3, 'c', 'b'}
```

Both union() and update() will exclude any duplicate items.

- intersection.update() metodu iki set'te yalnızca aynı olan elemanları alır ve yeni bir set içinde verir (AnB) :

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
x.intersection_update(y)
print(x) >>> {'apple'}
```

- symmetric\_difference\_update() metodu yalnızca ortak olmayan elemanları yazdırır (AnB)' :

```
x = {10, 20, 30, 40}
y = {40, 20, 30, 15}
```

```
x.symmetric_difference_update(y)
print(x) >>> {10, 15}
```

- `symmetric_difference()` metodu ise ortak olanlar hariç bütün elemanları yazdırır:

```
x = {10, 20, 30, 40, 12}
y = {40, 20, 30, 15}
print(x.symmetric_difference(y)) >>> {10, 12, 15}
```

- set'lerde pop kullanılmaz çünkü setler are unordered, so they do not have a strict pattern, you may not know which value will `pop()` erase.
- If we need an immutable set, we can create a frozen set with the `frozenset()` function. frozen setler bir nevi dondurulmuş kümelerdir, ekleme silme değiştirme yapılamaz. `frozenset()` metoduyla yapılır.

şeması:

```
my_set = frozenset(['a', 'b', 'c'])
This line creates a frozen set from a list.
```

- set comprehension:

```
a = {x for x in range(10) if x % 2 == 0}
```

örnek:

```
text = "When in the Course of human events it becomes necessary for one people..."
a = {ch.lower() for ch in text if ch.isalpha()}
```

- örnek:

```
Program to perform different set operations like in mathematics

define three sets
E = {0, 2, 4, 6, 8};
N = {1, 2, 3, 4, 5};

set union
print("Union of E and N is", E | N)

set intersection
print("Intersection of E and N is", E & N)

set difference
print("Difference of E and N is", E - N)

set symmetric difference
print("Symmetric difference of E and N is", E ^ N)
```

- $a = b$  a, b'ye eşittir demektir  
 $a \neq b$  a, b'ye eşit değildir demektir  
 $a \geq b$  a, b'den büyük eşittir  
 $a \leq b$  a, b'den küçük eşittir
- a b'den büyüktür True, False

```
a = int(input("a: "))
b = int(input("b: "))
result = a > b
print(f'a {a}, b {b} den büyüktür: {result}')
```

- final ve vize notları ortalama hesaplama:

```
vize1 = float(input("vize1 notu: "))
vize2 = float(input("vize2 notu: "))
final = float(input("final notu: "))
ortalama = (((vize1 + vize2) / 2) * 0.6) + (final * 0.4)
print(f'not ortalamanız: {ortalama} ve dersten geçme durumunuz {ortalama >= 50}')
```

yukarıdaki fonksiyonun lambda hali çok daha kolaydır:



```
ortalama = lambda vize, final: vize * 0.4 + final * 0.6
print(ortalama(40, 60))
```

- girilen sayının tek mi çift mi olduğunu yazdıran program:

```
def func(n):
 return "even" if n % 2 == 0 else "odd"
```

- sadece sondaki 0'ları silen program:

```
def func(n):
 if n == 0:
 return 0
 else:
 n = str(n)
 while n[-1] == "0":
 n = n[:-1]
 return int(n)
```

bir diğer yolu:

```
def func(n):
 try:
 return int(str(n).rstrip('0'))
 except ValueError:
 return 0
```

- verilen stringi tersten yazdıran program:

```
def func(n):
 return n[::-1]
```

- bir stringdeki sesli harfleri ! ile değiştiren program:

```
def func(n):
 return "".join(["!" if c in "aeiouAEIOU" else c for c in n])
print(func("merhaba")) >>> m!rh!b!
```

- girilen sayının pozitif mi negatif mi olma durumu:

```
sayi = int(input("sayiyi giriniz: "))
a = sayi >= 0
print(f'girdiğiniz sayi pozitif olma durumu: {a}')
```

- email ve sifre dogrulama:

```
email = "bugrakara@hotmail.com"
sifre = "bugrakara55"
girilenMail = input("eposta giriniz: ")
girilenSifre = input("sifre giriniz: ")
a = girilenMail == email
b = girilenSifre == sifre
print(f'girilenMailin dogru olma durumu: {a}')
```

- and operatoru soyle calisir:

```
if a > 0 and a != 0:
 girilen ifadede bir taraf bile yanlıssa sonuç yanlıştır.
```

- or operatoru soyle calisir:

```
if a == 2 or a != 5:
 eger ikisinden en az birisi doğruysa True verir. Her ikisi de yanlıssa False verir. A veya B gibi düşün.
```

bunlar iç içe de geçebilir:

```
result = ((x > 5) or (x < 10)) and (x % 2 == 0)
```

yani x 5'ten büyük veya 10'dan küçükse de doğru kabul edecek, x çift sayıysa da doğru kabul edecek. ama her ikisi de yanlıssa False kabul edecek.

- bir sayının 0'la 100 arasında olup olmadığını yazdırır:

```
x = float(input("bir sayı giriniz: "))
```

```
result = (x > 0) and (x < 100)
```

```
print(result) >>> atıyorum 34 girersen True yazdırır.
```

ama eğer bunu True/False olarak yazdırmak istemiyorsan koşul metodunu kullanabilirsin:

```
a = float(input("bir sayı giriniz: "))
```

```
b = (a > 0) and (a < 100)
```

```
if b:
```

```
 print("girilen sayı 0'la 100 arasındadır")
```

```
else:
```

```
 print("girilen sayı 0'la 100 arasında değildir")
```

## #conditions

- conditionslar sadece sağladığı durumlarda kodumuzun çalışmasını istediğimiz tekil durumlarda kullanılır. while'ın aksine bir döngü, sonsuzluk ve tekrerrür değil tekil bir execution vardır. 3 anahtar kelimesi vardır:

- o if
- o elif (else if)
- o else

1. if: bu anahtar kelimedenden sonra istediğimiz şartları yazıp sonuna mutlaka colon (:) koyarız ve altına gelen kod blogu mutlaka indented olmalı:

```
if a < 10:
 print("a 10dan küçük")
```

if satırına birden fazla koşul ekleyebilirsiniz. bunun için **and** ya da **or** kelimelerini kullanabilirsiniz:

```
if deneme > 10 and deneme < 20:
```

and ile yazdığınız if satırınının true olabilmesi için (altına yazdığınız kod blogunu çalıştırması için) iki şart da geçerli olmalıdır. biri bile yanlışsa hata verir

```
if deneme > 10 or deneme %2 == 0:
```

or ile yazdığınız if satırınının true olabilmesi için en az birinin geçerli olması yeterlidir.

2. elif: if satırından başka şartları sorgulatmak istiyorsak if'den sonra gelen tüm ifleri elif olarak yazarız. bunun istisnaları da vardır. yani if'li bir şart satırı yazdık, eğer bu şart sağlanmıyorsa varsa altında diğer elif şartlarını denir. onlar da çalışmıyorsa en son else'i execute eder. eğer şart True ise altındaki blogu çalıştırır, değilse bir alta geçer. yani elif şu demektir, eğer önceki conditions true değilse bu condition'u dene:

```
if a < 10:
 print("a 10dan küçük")
elif a > 10:
 print("a 10dan büyük")
```

3. else: if ve elif'lerin hiçbirinin çalışmadığı durumda else kodu çalışır:

```
if a < 10:
 print("a 10dan küçük")
elif a > 10:
 print("a 10dan büyük")
else:
 print("a 10a eşit")
```

eğer tek bir satırdan oluşuyorsa tek satırda da yazabilirsiniz:

```
if a < 10: print("a 10dan küçük")
elif a > 10: print("a 10dan büyük")
else: print("a 10a eşit")
```

- örnek:

```
a = int(input("bir sayı giriniz: "))
if (a > 0) and (a % 2 == 0):
 print("girdiğin sayı çift ve doğal sayıdır")
elif a == 0:
 print("girdiğin sayı 0dır, ne pozitif ne negatif")
elif a > 0 and a % 2 != 0:
 print("girdiğin sayı pozitif tek sayıdır")
else:
 print("girdiğin sayı negatif bir sayıdır")
```

- you don't have to use indent and state it in one line if you have only one statement:

```
if 10 > 4: print("True")
```

you can do the same with if/else statement:

```
print("A") if a > b else print("B")
```

- if'ten sonra değişkeni yazıp direkt : koymak o değişken True ise demektir.

örnek:

```
is_male = True
if is_male:
 print("you are a male") >>> you are a male
```

tek başına bir int, string ya da float yazarsan da sorun olmuyor.

- You can have if statements inside if statements, this is called nested if statements:

```
x = 41
if x > 10:
 print("Above ten,")
 if x > 20:
 print("and also above 20!")
 else:
 print("but not above 20.")
```

else'i burada if bloğunun altında (hizasında değil) yazmadığın için 10'dan küçük bir sayı yazarsan hata verir. 10'dan büyük ama 20'den küçük sayı yazarsan else satırı çalışır.

- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error:

```
a = 33
b = 200
if b > a:
 pass
```

- örnek:

```
age = int(input("giriniz: "))

if age < 18 :
 price = 10
elif age ≥ 18 and age < 50:
 price = 25
elif age ≥ 50:
 price = 50

print(f'yaşınız {age} olduğundan dolayı ödemeniz gereken ücret {price} $') >>>
giriniz: 49
yaşınız 49 olduğundan dolayı ödemeniz gereken ücret 25
```

- ehliyet sorgulama:

```
isim = input("isim: ")
yas = int(input("yas: "))
egitim = input("egitim: ")

if (yas ≥ 18) and (egitim == "lise" or egitim == "universite"):
 print("ehliyet alabilirsiniz.")
else:
 print("ehliyet alamazsınız.")
```

hangi nedenle ehliyet alıp alamadığını sorgulamak için if içinde if yazmamız gerekir. o da şöyle:

```
isim = input("isim: ")
yas = int(input("yas: "))
egitim = input("egitim: ")

if (yas ≥ 18):
 if (egitim == "lise" or egitim == "universite"):
 print("ehliyet alabilirsiniz.")
 else:
 print("ehliyet alamazsınız çünkü eğitiminiz yeterli değil.")
```

```
else:
 print("ehliyet alamazsiniz çünkü yaşıınız 18den küçük.")
```

- koşul ifadelerinde şöyle bir ayrım vardır. mesela bir sayının 0'dan büyük çift bir sayı olup olmadığını sorguluyorsun:

```
sayi = int(input("bir sayi giriniz: "))

if sayi > 0 :
 if sayi % 2 == 0:
 print("girilen sayi 0'dan büyük ve çift")
 else:
 print("girilen sayi 0'dan büyük ama çift değil")
else:
 print("girilen sayi 0'dan büyük değil")
```

alt alta iki tane if yazarsan bir koşulun altına bir alt koşul ve ona uygun bir print de yazdırmış olursun.

1. else'in anlamı şu, birinci if'i (yani > 0) sağlıyor ama ikinci if'i (% 2 == 0) sağlamıyor, dolayısıyla çift değil

2. else ise sayi sifirdan büyük değil anlamına geliyor

- fonksiyonlarla conditionları şöyle birleştirebilirsin:

```
def func():
 a = 5
 if a > 0: return "a 0'dan büyüktür"
 if a == 5: return "a 5'tir"
 if a < 0: return "a negatiftir"
```

## #loops

- there are 2 types of loops in python:

1. for
2. while

- python has 4 loop keywords:

1. for: for a in b satırı b'deki her bir ögeyi (a'yı) alır, seçer.

```
▪ list = [1,2,3,4,5]
 for i in list:
 print(i)
```

2. while: True olduğu müddetçe while altına yazdığın bloğu çalıştırır.

```
▪ a = 5
 while a < 10:
 print(a)
 a += 1
```

3. continue: görür görmez o ögeyi atlar ve kodu çalıştırmaya devam eder:

```
▪ a = "spotify"
 for i in a:
 if i == "f":
 continue
 print(i) >>> s p o t i y
```

4. break: görür görmez döngüden çıkarır.

```
▪ a = "spotify"
 for i in a:
 if i == "f":
 break
 print(i) >>> s p o t i
```

- for döngüsüyle bir string'i yazdırırsan bütün harfleri alt alta yazdırır.

```
str = "bugra"
for i in str:
 print(i)
```

- örnek:

```
soru = "Lütfen yaşınızı giriniz, "
soru += "eğer girdiyseniz q yazınız:"
while True:
 yaş = input(soru)
 if yaş == 'q':
 break
 else:
 print(f"siz {yaş} yaşındasınız.")
```

- continue operator commands the interpreter to terminate the current loop iteration. Then, the interpreter proceeds with the next iteration:

```
ilk_sayi = 0
while ilk_sayi < 20:
 ilk_sayi += 1
 if ilk_sayi % 2 == 0:
 continue
 print(ilk_sayi) > 1 3 5 7 9 11 13 15 17 19
```

print() içine yazacağın şeye dikkat et:

```
a = "merhaba"
for i in a:
 print(i) >>>
```

m  
e  
r

```
h
a
b
a
```

ya da

```
a = "merhaba"
for i in a:
 print(a) >>>
```

merhaba

...

merhaba

merhaba'nın içindeki indeks sayısı kadar merhaba yazdırdı

örnek:

```
for num in range(2,9):
 if not num % 2:
 continue
 print(num) >>> 3 5 7
```

continue örnek:

```
for i in (0,1,2,3,4,5,6):
 if i == 2 or i == 4:
 continue
 print(i) >>> 0 1 3 5 6
```

- örnek:

```
while True:
 print('Enter your age:')
 age = input()
 if age.isdecimal():
 break
 print('Please enter a number for your age.')
while True:
 print('Select a new password (letters and numbers only):')
 password = input()
 if password.isalnum():
 break
 print('Passwords can only have letters and numbers.')
```

- n'e kadar olan fibonacci sayılarını yazdıran program:

```
def fibo(n):
 result = []
 a, b = 0, 1
 while a < n:
 result.append(a)
 a, b = b, a + b
 return result
print(fibo(160)) >>> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

- range() metodu da for metodunun bir alt dalıdır.

```
for i in range(-5,5):
 print(i) >>> -5 -4 -3 -2 -1 0 1 2 3 4 5
```

The else keyword in a range loop specifies a block of code to be executed when the loop is finished:

```
for i in range(6):
 print(i)
else:
 print("Finally finished!") > 0 1 2 3 4 5 Finally finished!
```

- nested loops:

```
sıfatlar = ["güzel", "çirkin", "hızlı"]
isimler = ["ev", "araba", "oyuncak"]
for a in sıfatlar:
 for b in isimler:
 print(a, b) >>>
güzel ev
güzel araba
güzel oyuncak
çirkin ev
çirkin araba
çirkin oyuncak
hızlı ev
hızlı araba
hızlı oyuncak
```

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error:

```
for x in [0, 1, 2]:
 pass
```

- An "if-elif" block can exist without the else block at the end. However, "elif" cannot stand on its own without an "if" step before it. yani ilk if'den sonraki bütün hepsi elif en son da tekli bir else.
- sayı arttırmalı while örneği:

```
i = 0
while i < 10:
 print(i)
 i += 1 >>> 0 1 2 ... 9
```

1. i'ye başta 0 değerini atadık
  2. daha sonra 1 10'dan küçük olduğu müddetçe i'yi yazdır dedik
  3. ve her yazdırmadan sonra i += 1 metoduyla i'nin değerini 1 artırdık
  4. i 10'dan küçük olmayana kadar devam etti
- istediği cevabı alana kadar ısrarla soru soran program:

```
soru = "merhaba. bu kalem senin mi? "
soru += "evet ya da hayır de:" #1
cevap = " " #2
while cevap != "hayır" or "evet":
 cevap = input(soru) #3
 if cevap == "evet":
 print("teşekkürler")
 break
 if cevap == "hayır":
 print("olsun")
 break
```

1. soru str'sine ek bir str eklemek için += metodunu kullandık
  2. cevap isimli boş bir str atadık
  3. cevap değişkenini input ile soruya atadık
- With the break statement we can stop the loop even if the while condition is true:

```
i = 1
while i < 6:
 print(i)
 if i == 3:
 break
 i += 1 >>> 1 2 3
```



- With the else statement we can run a block of code once when the condition no longer is true:

```
i = 1
while i < 6:
 print(i)
 i += 1
else:
 print("and i is no longer less than 6") >>> 1 2 3 4 5 i and is no longer less than 6
```

- örnek:

```
for i in range(1,20):
 if i == 1:
 print("1st")
 elif i == 2:
 print("2nd")
 elif i == 3:
 print("3rd")
 else:
 print(f'{i}th') >>>
```

```
1st
2nd
3rd
4th
5th
```

- bir listedeki elemanları şartlı olarak başka listeye taşıma:

```
isimler = ["ali", "ece", "su", "tan", "buğrahan", "levent"]
kısa_isimler = []
for i in isimler:
 if len(i) < 4:
 kısa_isimler.append(i)
print(kısa_isimler) >>> sadece ali ece su ve tan'ı alır kısa_isimler listesine
bu kodun list comprehension'lı çok daha kısa hâli:
```

```
isimler = ["ali", "ece", "su", "tan", "buğrahan", "levent"]
kısa_isimler = [i for i in isimler if len(i) < 4]
print(kısa_isimler)
```

- list comprehension:

```
for i in list:
 print(i)
```

metodunun kısa yolu list comprehension'dır:

```
[print(i) for i in list]
```

List comprehension offers a shorter syntax when you create a new list based on the values of an existing list:

```
newlist = [expression for item in iterable if condition == True]
```

with list comprehension you can do all that in just one line:

```
isimler = ["kerem", "agah", "nihal", "bekir", "cüneyt", "bugra"]
yeniListe = [i for i in isimler if "a" in i]
print(yeniListe)
```

list comprehension + loop örnek:

```
list1 = [1,2,3,4,5,6]
list2 = [i*2 for i in list1]
print(list2) >>> [2, 4, 6, 8, 10, 12]
```

örnek2 (ilki list comprehension'sız uzun yol, 2.si list comprehension'lı kısa yol:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
new_list = []
for i in fruits:
```

```

 if "a" in i:
 new_list.append(i)
print(new_list)

```

ve

```

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist) >>> sadece içinde a olanları alır

```

örnek3:

```

even_nums = [x for x in range(21) if x%2 == 0]
print(even_nums) >>> [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

```

örnek4:

```

squares = [x*x for x in range(11)]
print(squares) >>> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

```

list comprehension with nested lists:

```

nums1 = [1, 2, 3]
nums2 = [4, 5, 6]
nums=[(x,y) for x in nums1 for y in nums2]
print(nums) >>> [(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]

```

örnek2:

```

harfler = ["a","b","c","d","e"]
siralama = ["birinci","ikinci","üçüncü","dördüncü","beşinci"]
harfSiralama = [(x, y) for x in harfler for y in siralama]
print(harfSiralama)

```

list comprehension with multiple if conditions:

```

nums = [x for x in range(21) if x%2==0 if x%5==0]
print(nums) >>> [0, 10, 20]

```

direkt art arda yazıyoruz aralarına and eklemeyen.

örnek2:

```

numbers = [i for i in range(1,101) if i % 5 == 0 if i % 6 == 0]
print(numbers) >>> [30, 60, 90]

```

bu kod fruits içinden elma harici (elma olmayanları) al ve newList'e at demektir.

```

newlist = [x for x in fruits if x != "apple"]

```

örnek:

```

numbers = [i for i in range(1,20) if i != 5 if i != 10 if i != 15]

```

örnek2:

```

newlist = [x for x in range(10) if x < 5]
print(newlist) >>> [0, 1, 2, 3, 4]

```

- bu

```

data = [[1, 2], [3, 4], [5, 6]]
output = []
for each_list in data:
 for element in each_list:
 output.append(element)

```

ve bu

```

data = [[1, 2], [3, 4], [5, 6]]
output = [element for each_list in data for element in each_list]

```

aynı anlama gelir.

- loop satırında istediğin modifikasyonu yapabilirsin:

```

isimler = ["bugra" , "ahmet" , "cemre"]
for a in isimler:
 print(f'benim adim {a}')
>>>
benim adim bugra
benim adim ahmet

```

benim adim cemre

- bir kullanıcının girdiği bir strindeki sesli harf sayısını yazdıran program:

```
sesliHarf = 'aeııoöüü'
sayaç = 0
kelime = input("bir kelime giriniz:")
for i in kelime:
 if i in sesliHarf:
 sayaç += 1
print(f"{kelime} kelimesinde tam {sayaç} sesli harf vardır.")
```

aynısının fonksiyon ile yazılmış hâli:

```
sesliHarf = 'aeııoöüü'
sayaç = 0
kelime = input("bir kelime giriniz:")
def seslidir(harf):
 return harf in sesliHarf
for harf in kelime:
 if seslidir(harf):
 sayaç += 1
print(f"{kelime} kelimesinde {sayaç} adet sesli harf var.")
```

örnek:

```
for i in [x**2 for x in range(5)]:
 print(i) >>> 0 1 4 9 16
```

- faktöriyel hesaplama:

```
sayi = int(input("bir sayi giriniz: "))
faktoriyel = 1 #1
for i in range(1, sayi + 1): #2
 faktoriyel = faktoriyel * i #3
print(faktoriyel)
```

1. faktöriyelde çarpmaya 1'den başladığı için initial value'yu 1'e eşitledik
2. sayının 1+'sını yazarak 1'den kendisine kadar tüm aralığı seçtik
3. bu sayıları çarptık

bunun math modülüyle çok daha kısa yolu:

```
import math
print(math.factorial(5)) >>> 120
```

- bir sayının rakamlarının faktöriyellerinin toplamının o sayıya eşit olup olmadığını veren program:

```
def func(number):
 nums = []
 for n in str(number):
 tot = 1
 for i in range(1,int(n)+1):
 tot *= i
 nums.append(tot)
 return True if number == sum(nums) else False
```

- girilen bir sayıya kadar olan toplamı bulma:

```
sayi = int(input("bir sayi giriniz: "))
toplam = 0
for i in range(1, sayi + 1):
 toplam = toplam + i
print(toplam)
```

- for döngüsünü liste değil de string bir ifade için kullanırsan tıpkı listenin elemanlarını alt alta yazdırdığı gibi stringin de harflerini alt alta yazdırır.

```
tuple = [(1,2), (3,4), (5,6)]
for i in tuple:
```

```
print(i) >>>
```

```
(1, 2)
```

```
(3, 4)
```

```
(5, 6)
```

iki farklı şey de atayabilirsin:

```
tuple = [(1,2), (3,4), (5,6)]
```

```
for a, b in tuple:
```

```
 print(a,b)
```

```
>>>
```

```
1 2
```

```
3 4
```

```
5 6
```

- dictionary'leri de for döngüsüyle yazdırabiliriz:

```
dictionary = {'k1' : 1, "k2" : 2, "k3" : 3}
```

```
for i in dictionary.items():
```

```
 print(i)
```

```
>>>
```

```
('k1', 1)
```

```
('k2', 2)
```

```
('k3', 3)
```

1. .values sadece value'ları yazdırır
2. .keys sadece keyleri yazdırır
3. .items hem keyleri hem value'ları yazdırır

- bir sayılar listesindeki sayıları for döngüsüyle toplama:

```
sayilar = [1, 3, 5, 7, 9, 12, 19, 21]
```

```
toplam = 0
```

```
for a in sayilar:
```

```
 toplam += a
```

```
print(toplam)
```

```
>>> 77
```

- o öncelikle 0'a eşit bir toplam değişkeni tanımlıyoruz.
- o daha sonra for döngüsüyle bütün elemanları a değişkeniyle topluyoruz ( += kullanarak)
- o ve finalde topladıklarımızı print'le yazdırıyoruz

- bir sayılar listesindeki tek olan sayıların karelerini yazdırma:

```
sayilar = [1, 3, 5, 7, 9, 12, 19, 21]
```

```
for a in sayilar:
```

```
 if (a % 2 == 1):
```

```
 print(a ** 2)
```

- while metodu bir denklem sağlandığı sürece o denklemi işlemlerini sağlar.

```
a = 0
```

```
while a < 100:
```

```
 print(a)
```

bu kod şöyle çalışır:

1. a'yı 0'a eşitleriz.
2. programa deriz ki, a, 100'den küçük olduğu sürece, a'yı (yani 0'ı) yazdır.
3. a bu süreçte değişmediği için sürekli a'yı yazdırır, a a a a a a a a a a a a a a a a a a a a ...

bu koda a += 1 satırını ekleyerek 1'den 100'e kadar sayıları yazdırmanın yolunu buluruz:

```
a = 0
```

```
while a < 100:
```

```
 a = a + 1
```

```
 print(a) >>> alt alta 1, 2, 3, ... , 99 şeklinde 100'e kadar yazdırır
```

işin garibi a = a + 1'i print(a) nın altına yazarsak 99a kadar yazdırıyor..

bunun daha kolay yolu:

```
for i in range (1,101):
 print(i)
```

- Kullanıcın girdiği iki sayı arasındaki sayıların toplamını gösteren python örneği:

```
toplam=0
sayi1=input('1. Sayı: ')
sayi2=input('2. Sayı: ')
for i in range(int(sayi1),int(sayi2)+1):
 toplam += i
print("{0} ile {1} arasındaki sayıların toplamı : {2}".format(sayi1,sayi2,toplam))
```

- kullanıcının girdiği sayıya kadar olan sayıları yazdırma:

```
sayi = int(input("bir sayı giriniz: "))
for i in range(1, int(sayi)+1):
 print(i)
```

- input yapmanın 2 yolu:

```
input("nasılsın:")
or
a = " "
a = input("nasılsın:")
```

- kullanıcı ismini girene kadar isim sorma:

```
name = ''
while not name:
 name = input("isminizi girin: ")
print(f'merhaba {name}')
```

" ifadesi False'a eşittir. not name demek boş olan name False'a eşit olduğundan True'ya denk gelir. while not name dediğimiz için alttaki satırı True olmayana (kullanıcı bir isim girene) kadar yazdırır.

- bir listeyi while döngüsüyle yazdırma:

```
arabalar = [1, 3, 40, 50, 100]
a = 0
while a < len(arabalar):
 print(arabalar[a])
 a += 1
```

for döngüsüyle yazdırma:

```
arabalar = [1, 3, 40, 50, 100]
for i in arabalar:
 print(i)
```

- kullanıcıdan alacağın 5 sayıyı ekrana sıralı bir şekilde yazdırma:

```
numbers = [] #1
i = 0 #2
while i < 5: #3
 sayi = int(input("sayiyi giriniz: ")) #4
 numbers.append(sayi) #5
 i += 1 #6
print(numbers)
>>>
sayiyi giriniz: 100
sayiyi giriniz: 200
sayiyi giriniz: 300
sayiyi giriniz: 400
sayiyi giriniz: 500
[100, 200, 300, 400, 500]
```

o numbers isminde içine inputla kullanıcıdan alacağımız verileri ekleyebileceğimiz bir liste tanımladık.

- o i'ye ilk 0 değeri verdik.
- o sonra i 5'ten küçük olduğu müddetçe (5 tane input versin diye, 10 istiyosan < 10) geçerli olan bir while döngüsü yazdık.
- o input ile soru prompt ettik
- o numbers.append(sayi) sayesinde başta boş olarak yarattığımız numbers listesinin içine inputları tanımladığımız "sayı" değişkenini tanımladık ki her eklenen append() metoduyla onun içine gitsin.
- o bu while döngüsü sonsuza kadar tekrar etmesin diye de i += 1 diyerek 1 artırdık (ta ki 5'e kadar)
- o daha sonra numbers.sort() metoduyla random girilen bu sayılar sıralanabilir.
- o liste halinde değil de yukarıdan aşağı string olarak yazdırmak istiyosan da finalde for döngüsü kullan.

sayısını kullanıcının belirlediği adet ürünü fiyat bilgisiyle beraber yazdırma:

```
urunler = []
adet = int(input('kaç ürün eklemek istiyorsunuz: '))
i = 0
while (i < adet):
 name = input('ürün ismi: ')
 price = input('ürün fiyatı: ')
 urunler.append({
 'name' : name,
 'price' : price
 })
 i += 1
for urun in urunler:
 print(f'urunun adı: {urun["name"]} urun fiyatı: {urun["price"]}')
>>>
kaç ürün eklemek istiyorsunuz: 3
ürün ismi: canga
ürün fiyatı: 1
ürün ismi: danette
ürün fiyatı: 2
ürün ismi: rokko
ürün fiyatı: 3
urunun adı: canga urun fiyatı: 1
urunun adı: danette urun fiyatı: 2
urunun adı: rokko urun fiyatı: 3
```

#### • örnek:

```
cevap = input("lütfen pizzanızda istediğiniz malzemeleri giriniz:")
malzeme = []
while True:
 if cevap == 'quit':
 break
 else:
 malzeme.append(cevap)
 cevap = input("lütfen pizzanızda istediğiniz malzemeleri giriniz:")
print(malzeme)
```

#### • örnek2:

```
a = int(input("please tell me your age:"))
while a < 10:
 print("ücretiniz 10 TL")
 break
while a ≥ 10 and a < 30:
 print("ücretiniz 30 TL")
```

```

 break
while a ≥ 30:
 print("Ücretiniz 50 TL")
 break

```

- while metoduyla bir listeden eleman silme:

```

liste = ["a", 1, "b", 1, 1, "c", 1, "d", 1, 1, "e", 1, 1]
while 1 in liste:
 liste.remove(1)
print(liste) >>> ['a', 'b', 'c', 'd', 'e']

```

while kullanmadan, sadece liste.remove(1) dersen ilk 1'i siler yalnızca.

- filling a dictionary with user input:

```

responses = { }
polling_active = True
while polling_active:
 name = input("What is your name?:")
 city = input("Where do you live?:")
 responses[name] = city #1
 prompt = input("Would you like to let another person respond? (y/n) ")
 if prompt == "n":
 polling_active = False
print("\n--- Poll Results ---")
for name, city in responses.items():
 print(name + " lives in " + city + ".")
 # print(f"{name} lives in {city}")

```

o bunun fonksiyonunu tam anlayamadım

- 1'den 100'e kadar sayıların toplamı:

```

x = 0
toplam = 0
while x ≤ 100:
 toplam += x
 x += 1
print(f'toplam: {toplam}') >>> 5050

```

- 1'den 100'e kadar olan tek sayıların toplamı:

```

x = 0
toplam = 0
while x ≤ 100:
 x += 1
 if x % 2 == 1:
 continue
 toplam += x
 x += 1
print(f'toplam: {toplam}')

```

çift sayı için de  $x \% 2 == 0$  yapıyosun.

- python2'de print fonksiyonu şöyle yazılıyordu:

```
print "this is my house"
```

farzı misal python 2.6+ kullanıyosan python3'teki parantezli print özelliğini kullanmak için şöyle demen gerekir:

```
from __future__ import print_function
```

- bir sayının bölenlerini bulan program:

```

a = int(input("lütfen bölenlerini bulmak istediğiniz sayıyı giriniz:"))
divisors_of_a = []
for i in range(1, a+1):
 if a % i == 0:

```

```
divisors_of_a.append(i)
print(divisors_of_a)
```

- palindrom olup olmadığını sorgulayan program:

```
a = input("enter the word:")
if a == a[::-1]:
 print("this is a polindrome.")
else:
 print("no this is not a polindrom")
```

- enumerate metodu bir stringdeki harfleri indeks indeks yazdırır:

```
index = 0
mesaj = "oda"
for harf in enumerate(mesaj):
 print(harf)
>>>
(0, 'o')
(1, 'd')
(2, 'a')
```

- bir diğer metod ise zip metodu, iki listenin elemanlarını birbiriyle ilintilemek, eşleştirmek için kullanılır:

```
liste1 = ["a", "b", "c", "d"]
liste2 = [10, 15, 20, 25]
a = list(zip(liste1, liste2))
print(a)
>>> [('a', 10), ('b', 15), ('c', 20), ('d', 25)]
```

- bir kelimenin harflerini bir liste halinde yazdırma:

```
myString = "merhaba"
myList = []
for harf in myString:
 myList.append(harf)
print(myList)
>>> ['m', 'e', 'r', 'h', 'a', 'b', 'a']
```

bunun aynısını çok daha basit bir şekilde list comprehensionla yazdırabiliriz:

```
myString = "merhaba"
myList = [harf for harf in myString]
print(myList)
>>> ['m', 'e', 'r', 'h', 'a', 'b', 'a']
```

- bir listedeki doğum tarihlerinden otomatik yaş hesaplama:

```
dogumTarihleri = [2001, 1995, 1969, 1973]
yaşlar = [2021 - yaş for yaş in dogumTarihleri]
print(yaşlar) >>> [20, 26, 52, 48]
```

- bir listeden bir listeye eleman atama:

```
liste1 = [1,2,3,4,5]
liste2 = [a for a in liste1]
print(liste2) >>> [1, 2, 3, 4, 5]
```

örnek:

```
liste1 = [1,2,3,4,5]
liste2 = ["a"*2 for a in liste1]
print(liste2) >>> ['aa', 'aa', 'aa', 'aa', 'aa']
```

- flag örnek:

```
prompt = "tell me your age."
prompt += " if you don't want to, type Q:"
active = True
while active:
 msg = input(prompt)
```



```

if msg == "Q":
 active = False
else:
 print(msg)

```

- isim sorma metodu:

```

isim = input("isminizi giriniz: ")
a = input(f"isminiz {isim} mu (E / H) ? : ")
if a == "E":
 print("dogrulandı, isminiz", isim)
if a == "H":
 print("Üzgünüm, değil.")
else:
 print("geçersiz bir değer girdiniz.")

```

- sinema/tiyatro ogrenci indirimini sorgulama:

```

"""Kullanıcıya sinema ya da tiyatro tercihi sorulsun. Sinema izlemek için 15 TL,
tiyatro için 10 TL ödenmesi gerekmektedir.
Öğrencilere %50 indirim yapıldığı düşünülerek öğrenci ise indirim yapılan;
öğrenci değilse indirimsiz tutarı hesaplayarak
ekrana yazdıran kodu yazınız."""
ogrenci_mi = input("ogrenci misiniz? (E/H) : ")
secim = input("sinema mı tiyatro mu? : ")
if ogrenci_mi == "E":
 if secim == "sinema":
 print("7.5 TL")
 elif secim == "tiyatro":
 print("10 TL")
if ogrenci_mi == "H":
 if secim == "tiyatro":
 print("15 TL")
 elif secim == "sinema":
 print("20 TL")

```

- girilen sayının asal olup olmadığını kontrol etme:

```

sayi = int(input("bir sayı giriniz: "))
asalmi = True #başlangıçta asal olarak tanımlayalım (asalmi = True)
if sayi == 1: #1 asal sayı şartını sağladığı ama asal olmadığı için ona özel istisnai
 bir if bloğu yazdık
 print("1 asal sayı değildir.")
for i in range(2, sayi): #2den itibaren alacağımız için 2'den girilen sayı'ya kadar
 olanları alıyoruz
 if sayi % i == 0: #girdiğimiz sayının 2'den kendisine kadar olan sayılara kalansız
 bölünüp bölünmediğini görmek için for döngüsü altında if bloğu kullandık.
 asalmi = False #eğer bölünebiliyorsa asalmi ifadesi artık False'tur ve döngüden
 çıkılır
 break
if asalmi == True: #ama döngüden çıkmayıp True olarak kalmaya devam ediyorsa artık sayı
 asaldır diyebiliriz
 print("sayı asal")
else:
 print("sayı asal değildir.") #bunların haricinde de asal değildir yazdırıyoruz

```

bir başka yolu:

```

sayac = 0
sayi = int(input('Sayı: '))
for i in range(2, int(sayi)):
 if(int(sayi)%i==0):

```

```
 sayac+=1
 break
if(sayac≠0):
 print("Sayı Asal Değil")
else:
 print("Sayı Asal")
```

for ve while ile else kullanımı çok nadirdir.

## #functions

- A function is a block of reusable code that is used to perform a specific action. The advantages of using functions are:
  - o Reducing duplication of code
  - o Decomposing complex problems into simpler pieces
  - o Improving clarity of the code
  - o Reuse of code
  - o Information hiding
  - o Functions in Python are first-class citizens. It means that functions have equal status with other objects in Python. Functions can be assigned to variables, stored in collections, or passed as arguments. This brings additional flexibility to the language. Functions in Python are objects (like almost everything else). They can be manipulated like other objects in Python. Therefore functions are called first-class citizens. This is not true in other OOP languages like Java or C#.
  - o Functions can be defined inside a module, a class, or another function. Function defined inside a class is called a method.
- bir fonksiyon def kelimesiyle tanımlanır (definition) ve fonksiyon blogunun altına yazacak kodlar bir indent boşluklu olmalıdır:

```
def fonksiyon():
 pass
```

alt bloguna iş (fonksiyon) ekleyebilirsiniz (ve daha sonra çağırarak gerçekleştirebilirsiniz):

```
def func():
 print("this is a function")
func() >>> this is a function
```

yukarıdaki ile şu aynı sonuca varır:

```
def func():
 return "hi"
print(func())
```

- her fonksiyonun kendine özel sayılardan oluşan bir ID'si vardır. Her seferinde değişir bu ID:

```
def f():
 print("bu bir fonksiyondur")
print(id(f)) >>> 2333074583616
```

- fonksiyonlara örnek:

```
def add(sayı):
 return sayı + 5
t = add(6)
print(t) >>> 11
```

- there are 3 kinds of functions:
  - o functions that are always available for usage, dir(), len(), abs()
  - o functions that are contained within external modules which must be imported
  - o functions defined by a programmer with the def keyword

1.

```
a = "12345678"
b = len(a)
print(b)
```

2.

```
from math import sqrt
def cube(x):
 return x * x * x
print(abs(-1))
print(cube(9))
print(sqrt(81))
```

3.

```
def f():
 print("bu bir fonksiyondur")
```

- fonksiyon parantezlerinin içine daha sonra kullanmak üzere bilgi de ekleyebilirsiniz:

```
def selamlama(name):
 print("merhaba", name)
selamlama("bugra") >>> merhaba buğra
```

örnek:

```
def favorite_book(title):
 print("One of my favorite books is", title)
favorite_book("Alice in Wonderland") >>> One of my favorite books is Alice in Wonderland
```

- how to pass a list as an argument to a function:

```
def motor(marka):
 for i in marka:
 print(f"{i} bir motordur.")
motor_isimleri = ["harley", "yamaha", "honda"]
motor(motor_isimleri) >>>
harley bir motordur.
yamaha bir motordur.
honda bir motordur.
```

örnek:

```
def func(songs):
 for song in songs:
 print(f"{song} is a beautiful song.")
songs_names = ["downtown", "rainbow", "offertorium"]
func(songs_names)
```

yukarıdaki örneklerin ana şeması:

```
def func(a):
 for i in a:
 print(i)
b = [1,2,3]
func(b)
```

- what is argument? An argument is a piece of information that is passed from a function call to a function.

you can pass arguments to your functions in a number of ways. Some:

- o positional arguments, which need to be in the same order the parameters were written. Yani argümanlar yazıldığı sırayla fonksiyona işleniyorsa positional arguments tekniği.
  - o keywords arguments, where each argument consists of a variable name and a value; and lists and dictionaries of values.
- positional argument örnek:

```
def evcil_hayvan(hayvan_türü, hayvanın_ismi):
 print("Evcil hayvanım bir", hayvan_türü, "adı da", hayvanın_ismi)
evcil_hayvan("köpek", "ronaldo")
```

positional argument'ta print içine yazdığın (a, b) sırası mühim değildir. mühim olan fonksiyonu yazdığın satırdaki sırayla (def ...) fonksiyonu bilahire çağırdığın kod satırındaki sıra aynı olması.

```
def evcil_hayvan(hayvan_türü, hayvanın_ismi):
 print("Evcil hayvanımın adı", hayvanın_ismi, "türü de", hayvan_türü)
evcil_hayvan("kedi", "tekir.") >>> Evcil hayvanımın adı tekir. türü de kedi
```

def fonksiyon(a, b) ve fonksiyon("c", "d") a=c ve b=d olmak üzere eşleştirecektir. diğer bir örnek:

```
def fav_game(game_name, game_type):
 print(f"my fav game is {game_name} and its a {game_type}")
fav_game("valorant", "mmo") >>> my fav game is valorant and its a mmo
```

fakat sırayı farklı yazarsan istemediğin sonuçlar alabilirsin:

```
def fav_game(game_name, game_type):
 print(f"my fav game is {game_name} and its a {game_type}")
fav_game("mmo", "valorant") >>> my fav game is mmo and its a valorant
```

bunun için keyword methodunu kullanmalısın. A keyword argument is a name-value pair that you pass to a function. You directly associate the name and the value within the argument, so when you pass the argument to the function, there's no confusion. işte bu uyumsuzluğun keywords argument metoduyla düzeltilmiş hâli:

```
def fav_game(game_name, game_type):
 print(f"my fav game is {game_name} and its a {game_type}")
fav_game(game_type="mmo", game_name="valorant")
```

örnek:

```
def araba(araba_markasi, araba_fiyati):
 print("\nMarkası", araba_markasi, "olan arabanın fiyatı", araba_fiyati)
 print("\nFiyatı", araba_fiyati, "olan arabanın markası", araba_markasi)
araba(araba_markasi="Mercedes", araba_fiyati=100.000) >>>
Markası mazda olan arabanın fiyatı 100000
Fiyatı 100000 olan arabanın markası mazda
```

The order of keyword arguments doesn't matter because Python knows where each value should go. The following two function calls are equivalent:

```
describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
```

- iç içe fonksiyonlar:

```
def outer(num1):
 print("outer")
 def inner_increment(num1):
 print("inner")
 return num1 + 5
 num2 = inner_increment(num1)
 print(num1, num2)
outer(10) >>> 10 15
```

burada yaptığımız işleme encapsulation deniyor.

class altında oluşturulan fonksiyonlara method denir.

a function as a method:

```
class Class():
 def func():
 print("func() method")
```

a function as a function:

```
def f():
 print("f() function")
```

a function as an inner function:

```
def g():
 def f():
 print("f() inner function")
```

remember that the () brackets in a function call means to execute it.

- default values: When writing a function, you can define a default value for each parameter. If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's default value (the one you defined in the def... line). So when you define a default value for a parameter, you can exclude the corresponding argument you'd usually write in the function call. Using default values can simplify your function calls and clarify the ways in which your functions are typically

used. For example, if you notice that most of the calls to `describe_pet()` are being used to describe dogs, you can set the default value of `animal_type` to 'dog'. Now anyone calling `describe_pet()` for a dog can omit that information:

```
def describe_pet(pet_name, animal_type='dog'):
 print("I have a " + animal_type + ".")
 print("My " + animal_type + "'s name is " + pet_name.title() + ".")
describe_pet(pet_name='willie') >>>
I have a dog.
My dog's name is Willie.
```

örnek:

```
def araba(araba_modeli, araba_ismi = "mazda"):
 print("arabamın ismi", araba_ismi, "ve modeli", araba_modeli)
araba(araba_modeli=2000) >>> arabamın ismi mazda ve modeli 2000
```

örnek2:

```
def vehicle(vehicle_make, vehicle_type = 'bike'):
 print(f"I use {vehicle_type} which is a {vehicle_make} brand.")
vehicle("harley") >>> I use bike which is a harley brand.
```

when you use default values, any parameter with a default value needs to be listed after all the parameters that don't have default values. This allows Python to continue interpreting positional arguments correctly.

- return keyword in python functions: a return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned. Note: Return statement can not be used outside the function.

şeması şu şekildedir:

```
def fun():
 statements
 .
 .
 return [expression]
```

örnek:

```
def add(a, b):
 # returning sum of a and b
 return a + b
def is_true(a):
 # returning boolean of a
 return bool(a)
calling function
res = add(2, 3)
print("Result of add function is { }".format(res))
res = is_true(2 < 5)
print("\nResult of is_true function is { }".format(res))
```

return önemi:

```
def cube(number):
 number*number*number
print(cube(4)) >>> None
```

bu kodda return demezsen None yazdıracaktır.

return kullanırsan print kullanmak zorundasın, printi yukarda kullanırsan sadece çağırman yeterlidir:

```
def yazılım():
 print("yazılım öğreniyorum")
yazılım()
ile
```

```
def yazılım():
 return "yazılım öğreniyorum"
print(yazılım())
```

aynı sonucu verir.

örnek2:

```
def cube(x):
 return x * x * x
print(cube(3)) >>> 27
```

a'nın b. kuvveti fonksiyonu:

```
def power(a, b):
 return a ** b
print(power(2, 5)) >>> 32
```

in python you can return multiple values from a function in 4 ways:

- o using object
- o using tuple
- o using list
- o using dictionary

1. using object: this is similar to C/C++ and Java, we can create a class (in C, struct) to hold multiple values and return an object of the class:

```
A Python program to return multiple
values from a method using class
class Program:
 def __init__(self):
 self.programName = "cpp"
 self.programHardness = "7/10"

def myProgram():
 return Program()

a = myProgram()

print(a.programName)
print(a.programHardness)
```

2. Using Tuple: A Tuple is a comma separated sequence of items. It is created with or without ():

```
This function returns a tuple
def fun():
 str = "geeksforgeeks"
 x = 20
 return str, x; # Return tuple, we could also
 # write (str, x)

Driver code to test above method
str, x = fun() # Assign returned tuple
print(str)
print(x) >>>
geeksforgeeks
20
```

3. Using a list: A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they are mutable:

```
A Python program to return multiple
values from a method using list
This function returns a list
def fun():
```

```

 str = "geeksforgeeks"
 x = 20
 return [str, x];
Driver code to test above method
list = fun()
print(list) >>> ['geeksforgeeks', 20]

```

4. Using a Dictionary: A Dictionary is similar to hash or map in other languages:

```

A Python program to return multiple
values from a method using dictionary
This function returns a dictionary
def fun():
 d = dict();
 d['str'] = "GeeksforGeeks"
 d['x'] = 20
 return d
Driver code to test above method
d = fun()
print(d) >>> {'str': 'GeeksforGeeks', 'x': 20}

```

- normalde def... satırına yazdığın argüman kadar fonksiyon çağırdığın satırda argüman girebilirsin, yoksa hata verir:

```

def func(a, b):
 pass
func("x", "y")

```

if you don't know how many arguments will be passed, you may use \*args method:

```

def func(*num):
 return sum(num)
print(add_nums(50, 50, 110 #...)) >>> 210

```

yani sayısi belirsiz argüman kullanırken argümanın başına \* atman yeterli. daha sonra fonksiyonu çağırırken istediğin kadar parametre kullanabilirsin. Arbitrary Arguments are often shortened to \*args but it may be any word.

örnek:

```

def argüman_sayisi(*args):
 r = 0
 for i in args:
 r += 1
 return r
print(argüman_sayisi("a","a")) >>> 2

```

örnek:

```

def sandwich(*materials):
 print("your sandwich has: ")
 for i in materials:
 print(i)
sandwich("cheese", "pepper")
sandwich("cheese", "pepper", "salt")
sandwich("cheese", "pepper", "salt", "sausage")

```

- The terms parameter and argument can be used for the same thing: information that are passed into a function. From a function's perspective:
  - o A parameter is the variable listed inside the parentheses in the function definition.
  - o An argument is the value that is sent to the function when it is called.
- you may also use key = value syntax to simplify things:

```

def personae(p1, p2, p3):
 print("his name is", p2)

```



```
personae(p1 = "wolff", p2 = "albrecht", p3 = "hermann") >>>
his name is albrecht
```

this method is called keyword arguments. The phrase Keyword Arguments are often shortened to kwargs in Python documentations.

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly:

```
def show(**bilgiler):
 for i in bilgiler:
 print(f"{i}: {bilgiler[i]}")
show(isim="bugra",soyisim="kara",yaş=19,cinsiyet="erkek")
>>>
isim: bugra
soyisim: kara
yaş: 19
cinsiyet: erkek
```

arbitrary Keyword Arguments are often shortened to \*\*kwargs in Python documentations.

- daha iyi anlamak açısından positional/keywords arguments ayrımı:  
this is positional

```
def make_shirt(size, text):
 print("the size of this shirt is", size, "and the text is", text)
make_shirt(34, "beHappy") >>> the size of this shirt is 34 and the text is beHappy
this is keywords
```

```
def make_shirt(text, size):
 print("the size of this shirt is", size, "and the text is", text)
make_shirt(size = 34, text = "beHappy") >>> the size of this shirt is 34 and the text is beHappy
```

text&size yerlerini değiştirdik ama tanımlı oldukları için sorun çıkmadı.

- python'da ctrl + / ile seçili satırları yorum satırı yapabilirsin. vs code'da ctrl K + ctrl C ile yorum satırı shift+alt+a ile de "" satırı.
- making a parameter optional: eğer fonksiyon satırında eşittir ile beforehand bir değer atarsan parametreye daha sonra fonksiyonu çağırdığında argüman vermesen de o yazdığın default değeri hesaba katar (verirsen onu katar):

```
def func(x=3):
 return x**2
print(func()) >>> 9
```

örnek:

```
def tam_isim(ilk_isim, ikinci_isim, üçüncü_isim):
 a = ilk_isim + " " + ikinci_isim + " " + üçüncü_isim
 return a
print(tam_isim("ali", "mehmet", "celâl")) >>> ali mehmet celâl
```

bu fonksiyonda çağırırken en az üç argüman (ali,mehmet,celal) vermek zorundasın, yoksa error verir. fakat bazen üçüncü\_isim girmek istemeyebilirsin, bu durumda üçüncü\_isim'i optional yapmalısın. bunun için de " " default value atarsın:

```
def tam_isim(ilk_isim, ikinci_isim, üçüncü_isim = " "):
 if üçüncü_isim:
 a = ilk_isim + " " + ikinci_isim + " " + üçüncü_isim
 else:
 a = ilk_isim + " " + ikinci_isim
 return a
print(tam_isim("ali", "mehmet")) >>> ali mehmet
```

burada üçüncü\_isim'i optional yaptık. yani eğer 3 tane argüman girersen 3 tanesini, 2 tane girersen sadece girdiğin 2 tanesini yazdırır. this is called optional argument. you must place optionals at the end of the parameter.

örnek2:

```
def func(word, freq=3):
 return word * freq
print(func("bugra")) >>> bugrabugrabugra
```

- Python interprets non-empty strings as True. so if a: means if a is not empty.
- dikdörtgenin alanını hesaplama:

```
def dikdortgen_alan(genişlik, yükseklik):
 alan = float(genişlik) * float(yükseklik)
 print("dikdörtgenin alanı:", alan)
 return alan
gen = input("genişlik:")
yük = input("yükseklik:")
dikdortgen_alan(gen, yük)
```

dairenin çevresini hesaplama:

```
def daire_cevre(pi, r):
 cevre = 2 * pi * r
 print(f"dairenin çevresi {cevre}")
 return cevre
pi = float(input("pi'yi giriniz:"))
r = int(input("r'yi giriniz:"))
print(daire_cevre(pi, r))
```

silindirin hacmini hesaplama:

```
def silindirin_hacmi(r, h, pi):
 hacim = 2 * pi * r * h
 print(f"silindirin hacmi {hacim}")
 return hacim
pi = float(input("pi:"))
r = int(input("r:"))
h = int(input("h:"))
print(silindirin_hacmi(h, pi, r))
```

- verilen üç uzunluğun üçgen oluşturup oluşturamayacağını veren program:

```
def is_triangle(a, b, c):
 return a + b > c and b + c > a and a + c > b
```

- eğer bir fonksiyon blogunda başka bir fonksiyonun adını yazarsan o fonksiyonu çağırdığında içine yazdığın fonksiyonun işlemleri de yazdırılır:

```
def a():
 print("this is a function")
 b()

def b():
 print("this is b function")

a() >>>
this is a function
this is b function
```

- you can use functions with dictionaries:

```
def fonksiyon(isim, soyisim):
 kişi = {'isim': isim, 'soyisim': soyisim}
 return kişi
a = fonksiyon("mehmet", "akif")
print(a) >>> {'isim': 'mehmet', 'soyisim': 'akif'}
```

- use function-dictionary with an optional argument:

```
def fonksiyon(isim, soyisim, yaş = " "):
 kişi = {'isim': isim, 'soyisim': soyisim}
```

```

 if yaş:
 kişi['yaş'] = yaş
 return kişi
a = fonksiyon("mehmet", "akif")
b = fonksiyon("mehmet", "akif", 23)
print(a)
print(b) >>>
{'isim': 'mehmet', 'soyisim': 'akif', 'yaş': ' '}
{'isim': 'mehmet', 'soyisim': 'akif', 'yaş': 23}

```

örnek2:

```

def greet_users(names):
 for name in names:
 msg = "Hello, " + name.title() + "!"
 print(msg)
usernames = ['hannah', 'tyler', 'margot']
greet_users(usernames) >>>
Hello, Hannah!
Hello, Tyler!
Hello, Margot!

```

- return şeması şu şekildedir:

```

def func(a, b):
 msg = f"a is b"
 return msg
x = func("a","b")
print(x)

```

örnek:

```

def bilgiler(ülke, şehir):
 isim = şehir + " in " + ülke
 return isim.upper()
a = bilgiler("paris", "fransa")
print(a) >>> FRANSA IN PARIS

```

- fonksiyonlarda returnden sonra sözlük halinde bir condition yazıp [parametre] ismi verebiliyoruz:

```

def func(param1, param2):
 return {"abc": param2*2}[param1]
print(func("abc","deneme"))

```

eğer param1 olarak "abc" girilirse param2'yi 2 ile çarp demiş olduk burada.

- örnek:

```

def car(manufacturer, model, **color):
 car_dictionary = {
 'manufacturer' : manufacturer,
 'model' : model,
 }
 for color, value in color.items():
 car_dictionary[color] = value
 return car_dictionary
car1 = car('honda', 'civic', color = 'gray', year = 2015, owner = '2nd owner')
print(car1) >>>

```

burada şunu yaptık:

- o önce 3 argümanlı, car isminde bir fonksiyon tanımladık, fakat belki daha fazla argüman ekleriz diye son argümanın başına \*\* ekledik.
- o fonksiyonda dictionary kullanmak istediğimiz için farklı bir isimde dictionary tanımladık.
- o bu dictionary'e girdiğimiz ilk iki argüman'ın karşılıklarını atadık.

- o daha sonra loop döngüsüyle color argümanının key, value metodu uyguladık, sonuna .items() dedik.
- o return metoduyla dictionary'i fonksiyona döndürdük.
- o daha sonra fonksiyonu car1 isminde bir şeyle yazdırdık, içine istediğimiz şeyleri (ki \*\* dediğimiz için sınırsız sayıda yeni argument ekleyebiliriz) ekledik.
- o finalde de print metoduyla car1'i yazdırdık
- A parameter is the variable listed inside the parentheses in the function definition. An argument is the value that is sent to the function when it is called.
- fonksiyonları döngülerle de kullanabilirsin:

```
def my_function(food):
 for x in food:
 print(x)

fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

- fonksiyonlarla sayıları toplama:

```
def add_numbers(n1, n2):
 result = n1 + n2
 print("the sum is:", result)

n1 = 3.2
n2 = 6.5
add_numbers(n1, n2) >>> the sum is: 9.7
```

aynısını print yerine return metoduyla da yapabiliriz:

```
def add_numbers(n1, n2):
 result = n1 + n2
 return result

number1 = 3.2
number2 = 6.5
add_numbers(number1, number2)
result = add_numbers(number1, number2)
print("the sum is:", result) >>> the sum is: 9.7
```

- içini doldurabilirsin:

```
def deneme(isim):
 print("merhaba " + isim)

deneme("bugra")
deneme("cemre")

>>>
merhaba bugra
merhaba cemre
```

eğer deneme() yazıp boş bırakırsan hata verir. bunun sebebi deneme'nin içine bir parametre göndermemiş olmasıdır. eğer def deneme'nin içine bir default value atarsan deneme()'lar otomatik olarak onu yazdıracaktır içine başka bir string eklemediğin sürece:

```
def deneme(isim = 'kullanıcıİsmi'):
 print("merhaba " + isim)

deneme("bugra")
deneme("cemre")
deneme()

>>>
merhaba bugra
merhaba cemre
merhaba kullanıcıİsmi
```

- sayılar için return kullanımı:

```
def total(num1, num2):
 return num1 + num2

a = total(10, 25) #altına gidip CTRL + D ile 10, 300 yazarsan en son yazdığını işleme
```

```
alip 310 yazdırır.
```

```
print(a) >>> 35
```

- bir listedeki sayıların karelerini hesaplayıp toplayan program:

```
def square_sum(numbers):
 return sum([i**2 for i in numbers])
```

- mesela belirli bir durum doğruysa True döndür yanlışsa False döndür deniliyorsa bir fonksiyonda sadece True döndüren satırı yazman yeterlidir:

```
def func(a):
 return "a is True"
```

çünkü False ise zaten bir şey döndüremeyecektir.

- bir arrayde bir değerin olup olmadığını veren program:

```
def func(a, d):
 return d in a
```

- yas hesaplama:

```
def yasHesapla(dogumYili):
 return 2020 - dogumYili
ageBugra = yasHesapla(2000)
ageCemre = yasHesapla(2011)
ageAhmet = yasHesapla(1990)
print(ageAhmet, ageCemre, ageBugra) >>> 30 9 20
```

- bir baska ornek:

```
def change(i):
 i[0] = "samsun"

sehirler = ["ankara", "istanbul", "izmir"]

change(sehirler)
print(sehirler)

>>> ['samsun', 'istanbul', 'izmir']
```

kodu 3 farklı parça olarak düşün

- o parça, işlevi uygulandığı şeyin (i) 0. indeksini samsun ile değiştirmek olan bir fonksiyon
  - o parçada normal sehirler isminde bir liste tanımladık
  - o parçada ise change fonksiyonunu çağırıp sehirlere uyguladık
- fonksiyonlarla 2 parametre toplama:

```
def add(a, b):
 return sum((a, b))
print(add(30, 70)) >>> 100 #eğer 3 parametre toplamaya çalışırsan hata verir.
```

bunu 3 parametrelili yapmak istiyorsan:

```
def add(a, b, c):
 return sum((a, b, c))
print(add(30, 70, 33))
```

hem 2 hem 3 parametreliliyi çalıştıran hali:

```
def add(a, b, c = 0):
 return sum((a, b, c))
print(add(30, 10)) >>> 40
30, 10, 50 yazarsan 90 verecektir. 4+ yazarsan hata verir.
#bunları c = 0, d = 0, e = 0 şeklinde (return sum())'a eklemek kaydiyle) devam ettirebilirsin
```

- When you pass a list to a function, the function can modify the list. Any changes made to the list inside the function's body are permanent, allowing you to work efficiently even when you're dealing with large amounts of data.
- gönderilen bir kelimeyi belirtilen adet kadar ekranda yazdıran fonksiyon kodu:

```
def yazdir(kelime, adet):
 print(kelime * adet)
yazdir("abc\n", 5) >>> alt alta 5 kere abc yazdırır
```

- fonksiyon kullanarak bir sayının karesini alma:

```
def square(num):
 return num ** 2
a = square(12)
print(a) >>> 144
```

- map() function applies a given function to each item of an iterable (list, tuple etc.) and returns a list of the results. şeması şu şekildedir:

```
map(fonksiyonun_ismi, listenin_ismi)
```

mantığı şöyledir: mesela sayılardan oluşan bir liste var elinde ve bu listedeki her sayının karesini almak istiyorsun. bunun için bir fonksiyon yazdın:

```
def square(num):
 return num * num
```

listemiz de şu olsun:

```
my_list = [2,3,4,5,6,7,8,9]
```

bu listedeki her bir elemanı square() fonksiyonuna tabi tutmak için map() metodunu kullanıyorsun:

```
def square(n):
 return n*n
my_list = [2,3,4,5,6,7,8,9]
updated_list = map(square, my_list)
print(list(updated_list)) >>> [4, 9, 16, 25, 36, 49, 64, 81]
```

map() fonksiyonu bir built-in fonksiyondur ve başka built-in fonksiyonlarla kullanılabilir. mesela ondalıklı sayı yuvarlamaya yarayan round() fonksiyonuyla kullanalım:

```
my_list = [2.6743,3.63526,4.2325,5.9687967,6.3265,7.6988,8.232,9.6907]
updated_list = map(round, my_list)
print(list(updated_list)) >>> [3, 4, 4, 6, 6, 8, 8, 10]
```

stringlerle kullanımı:

```
def upper(a):
 return a.upper() #1
list = ["bugra","kara","samsun"] #2
updated_list = map(upper, list) #3
for i in updated_list:
 print(i, end= " ")#4 >>> BUGRA KARA SAMSUN
```

- o argümanını upper eden bir fonksiyon tanımladık ve return ile döndürdük
- o upper edilsin istediğimiz bir liste tanımladık
- o map metodu ile upper fonksiyonunu bu listeye uyguladık → map(fonksiyonİsmi, listeİsmi) (notice that we do not use brackets after functionName)
- o loop ile sonucu yazdırdık

sayılarla kullanımı:

```
def func(n):
 return n*10
liste = [1, 2, 3, 4]
updated_list = map(func, liste)
print(list(updated_list)) >>> [10, 20, 30, 40]
```

tuplelarla kullanımı:

```
def myMapFunc(n):
 return n.upper()
my_tuple = ('php','java','python','c++','c')
updated_list = map(myMapFunc, my_tuple)
print(list(updated_list)) >>> ['PHP', 'JAVA', 'PYTHON', 'C++', 'C']
```

for metoduyla da map() metoduyla da aynı işi yapabilirsin. önce for metoduyla yapılmış bir örnek:

```
list = [1,2,3,4,5,6]

def func(x):
 return x**x

newList = []
for i in list:
 newList.append(func(i))

print(newList)
```

bu da map() hâli:

```
def func(x):
 return x**x

liste = [1,2,3,4,5,6]

print(list(map(func, liste)))
```

ya da list comprehension da kullanabilirsin:

```
def func(x):
 return x**x

liste = [1,2,3,4,5,6]

print([func(x) for x in liste])
```

- filter() function discards elements of a sequence based on some criteria:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
nums_filtered = list(filter(lambda x: x % 2 == 0, nums))
print(nums_filtered) >>> [2, 4, 6, 8, 10, 12]
```

örnek:

```
def add(x):
 return x + 3

def isOdd(x):
 return x % 2 != 0

a = [1,2,3,4,5,6,7,8]
b = list(filter(isOdd,a))

print(b) >>> [1, 3, 5, 7]
```

- örnek:

```
names = ["ali", "enes", "buğrahan"]

def long_name(name):
 return len(name) > 5

print(list(filter(long_name, names))) >>> ['buğrahan']
```

- 

map() + filter() kullanımı:

```
def add(x):
 return x + 3

def isOdd(x):
```

```
return x % 2 != 0
```

```
a = [1,2,3,4,5,6,7,8]
```

```
c = list(map(add, filter(isOdd, a)))
```

```
print(c) >>> [4, 6, 8, 10]
```

- **lambda function:** it is possible to create anonymous functions in Python. Anonymous functions do not have a name. With the lambda keyword, little anonymous functions can be created. Anonymous functions are also called lambda functions by Python programmers. Lambda functions are restricted to a single expression. the lambda is slower than the list comprehension They can be used wherever normal functions can be used. şeması şu şekildedir:

```
func = lambda arg : arg + 2
```

```
print(func(3))
```

anlamı da:

```
variable = lambda arguments: operation
```

yukarıdaki lambda fonksiyonunun normal hali:

```
def func(arg):
```

```
 return arg + 2
```

```
print(func(3))
```

a lambda function can take any number of arguments, but can only have one expression:

```
x = lambda a : a + 10
```

```
print(x(20)) >>> 30
```

lambda example:

```
x = lambda a, b, c : a + b + c
```

```
print(x(5, 6, 2)) >>> 13
```

örnek:

```
y = 6
```

```
z = lambda x: x * y
```

```
print(z(8))
```

lambda + normal fonksiyon kullanımı:

```
def myfunc(n):
```

```
 return lambda a : a * n
```

```
mytripler = myfunc(3)
```

```
print(mytripler(11)) >>> 33
```

lambda + map() kullanımı:

```
nums = [1,2,3]
```

```
sqr_nums = map(lambda a:a**2, nums)
```

```
print(list(sqr_nums)) >>> [1, 4, 9]
```

Use lambda functions when an anonymous function is required for a short period of time.

örnek2:

```
x = map(lambda e: e.upper(), ["one","two","three"])
```

```
print(list(x))
```

- fonksiyonların içinde atadığın değişkenler başka yerdeki değişkenleri değiştirmez:

```
a = 5
```

```
def fonksiyon():
```

```
 a = 10
```

```
fonksiyon()
```

```
print(a) >>> 5
```

örnek:

```
isim = "bugra"
```

```
def isimDegistir(yeni_isim):
```

```
 isim = yeni_isim
```



```
print(isim)
isimDegistir("cemre") >>> cemre
```

- how do you code this?

```
[(0,0),(1,1),(2,2),(3,3)]
```

answer:

```
list = []
for a in range(4):
 for b in range(4):
 if a == b:
 list.append((a,b))
print(list)
```

fakat bunun daha kolay yolu var, o da lambda metoduyla:

```
list = list((a,b) for a in range(4) for b in range(4) if a == b)
print(list)
```

- decorative functions: dekoratif fonksiyonları bir fonksiyona özellik eklemek istediğimizde kullanıyoruz. bir özelliği birkaç yerde kullanmak istiyorsak bunun için bir decorative function oluşturup bu fonksiyonu birçok farklı fonksiyonda kullanabiliyoruz:

```
def decorator_deneme(function_deneme): #1
 def wrapper(): #2
 print("fonksiyondan önceki işlemler")
 function_deneme() #3
 print("fonksiyondan sonraki işlemler")
 return wrapper #4
def hello(): #5
 print("hello")
def merhaba():
 print("merhaba")
hello = decorator_deneme(hello)
hello()
print("*" * 10)
merhaba = decorator_deneme(merhaba)
merhaba() >>>
fonksiyondan önceki işlemler
hello
fonksiyondan sonraki işlemler

fonksiyondan önceki işlemler
merhaba
fonksiyondan sonraki işlemler
```

1. decorator\_deneme isminde bir fonksiyon yazdık bu fonksiyon bizden function\_deneme adında bir fonksiyon bekliyor.
2. wrapper() isminde bir inner function yazdık
3. function\_denemeyi çağırıyoruz ve önce ve sonra olmak üzere işlemler yazıyoruz
4. return metoduyla iç fonksiyonu döndürüyoruz
5. istediğimiz şey hello() fonksiyonunu çağırdığımızda dolaylı olarak wrapper() blogundaki işlemler de tetiklensin.

bunun yerine @decorator metoduyla birbiri içine atmadan direkt çağırabiliriz:

```
@decorator_deneme
def hello():
 print("hello")
hello() >>>
fonksiyondan önceki işlemler
hello
fonksiyondan sonraki işlemler
```

örnek:

```
import math
import time
def calculate_time(func):
 def inner(*args,**kwargs):
 start = time.time()
 time.sleep(1)
 func(*args,**kwargs)
 finish = time.time()
 print("fonksiyon "+func.__name__ +" " + str(finish-start) + " saniye sürdü.")
 return inner
@calculate_time
def usalma(a,b):
 print("üs sonucu:", math.pow(a,b))
@calculate_time
def faktoriyel(num):
 print("faktöriyel sonucu:", math.factorial(num))
@calculate_time
def toplama(a,b):
 print("toplama sonucu:", a+b)
usalma(2,3)
faktoriyel(4)
toplama(10,20) >>>
üs sonucu: 8.0
fonksiyon usalma 1.0144166946411133 saniye sürdü.
faktöriyel sonucu: 24
fonksiyon faktoriyel 1.0029151439666748 saniye sürdü.
toplama sonucu: 30
fonksiyon toplama 1.0036957263946533 saniye sürdü.
```

- decorator örnek:

```
class Giriş():
 def __init__(self, mesaj = "Müşteri Numaranız:"):
 cevap = input(mesaj)
 print("Hoş Geldiniz { }!".format(cevap))
 @classmethod
 def paroladan(cls):
 mesaj = "lütfen parolanızı giriniz:"
 cls(mesaj)
 @classmethod
 def tcknden(cls):
 mesaj = "lütfen tc'nizi giriniz:"
 cls(mesaj)
Giriş() >>> müşteri numaranız: 123123123 >>> Hoş Geldiniz 123123123!
```

## #arrays

- An array is a data structure that stores values of same data type. In Python, this is the main difference between arrays and lists. yani listeler birden çok türde veriyi içerebilirken arrayler yalnızca tek bir tip veriyi barındırabilir. array oluşturmak için
  - o array modülünü import etmek
  - o array.array() metodunu kullanmak gerekiyor.

arraylerin şeması şöyledir:

```
import array
arrayName = array(typecode, [elements])
```

- 
- örnek:

```
import array
sample_array = array.array('i', [1, 2, 3])
for i in sample_array:
 print(i)
print(type(sample_array))
```

- o burada i diyerek type code belirledik.

- Arrays are used to store multiple values in one single variable:

```
cars = ["Ford", "Volvo", "BMW"]
```

- you can print items by index (print(cars[0]) or change items (cars[0]="Toyota")

- you can use:

- o for
- o append()
- o pop()
- o remove()
- o insert()
- o index()
- o extend()
- o count()
- o reverse()
- o count()
- o sort()
- o fromlist()etc. methods with arrays.

- UML: unified modelling language.

- çift katlardaki elemanları silen program:

```
def func(n):
 return n[::-2]
print(func([1, 2, 3, 4, 5, 6])) >>> [1, 3, 5]
```

- bir array'in elemanlarını birbirleriyle çarpma:

```
def func(n):
 a = 1
 for i in n:
 a *= i
 return a
```

- girdiğin sayıya kadar array yazdıran program:

```
def func(n):
 return [i for i in range(1,n+1)]
print(func(5)) >>> [1,2,3,4,5]
```

- arrayin elemanlarını toplayan program:

```
def func(arr):
 return sum([i for i in arr])
```

- array'deki minimum sayıyı bulan program:

```
def func(n):
 return min(n)
```

## 2. yolu

```
def func(n):
 smallest = n[0]
 for i in n:
 if i < smallest:
 smallest = i
 return smallest
```

- fromlist() metodu bir listeden array'a eleman eklemeni sağlar:

```
import array
my_array = array.array('i', [1,2,3,4,5])
c=[11,12,13]
my_array.fromlist(c)
print(list(my_array)) >>> [1, 2, 3, 4, 5, 11, 12, 13]
```

-

## #OOP

- there are 3 main programming paradigms:
  - o procedural programming
  - o functional programming
  - o OOP programming:

OOP yani object-oriented programming uses objects and their interactions to design applications and computer programs. In OOP you write classes that represent real-world things and situations, and create objects based on these classes. Python is an oop language. Almost everything in Python is an object, with its properties and methods. Integers, strings, lists, dictionaries, tuples, functions, and modules are Python objects. A Class is an object but not built-in like these but a user-defined object. When you create a class, you define the general behavior that a whole category of objects can have. Making an object from a class is called instantiation, and you work with instances of a class. yani object'nin diğer adı instance'tır.in Python, a class needs to be instantiated before use. shortly a class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state. If Car is a class, then Audi A6 is an object of the Car class and def drive() function is a method under that class.

syntax of OOP:

```
class Hello:
 pass
```

- Inside a class, we can define attributes and methods. An attribute is a characteristic, a feature of an object. This can be for example a salary of an employee. A method is a function under the class blog that we can perform certain deeds. Technically, attributes are variables and methods are functions defined inside a class. 2 çeşit attribute vardır:
  - o class attribute
  - o instance attribute

1. class attribute: they have the same value for all class instances. init()'in altında yazılmazlar.

```
class Dog:
 name = "aaa"
```

class attribute atarken alelade bir değişken ataması yapar gibi yapıyoruz.

class attribute'ları yazdırmak için class bloguna print metodu ekleriz (sonrasında sınıfı çağırmamıza gerek kalmadan):

```
class Animal:
 tür = "köpek"
 print(tür) >>> köpek
```

class attribute'lara value eklemenin bir yolu append metodudur:

```
class Kişi:
 yetenekler = []
ahmet = Kişi()
ahmet.yetenekler.append("arabaSürme")
print(Kişi.yetenekler) >>> ['arabaSürme']
```

2. instance attribute: yaparken ise \_\_init\_\_() ve self metotlarından yararlanıyoruz. buradaki 'self.name = name' instance attribute'tur init()'in altında olduğu için. An instance attribute's value is specific to a particular instance of the class.

```
class Dog:
 def __init__(self):
 self.name = "aaa"
```

use class attributes to define properties that should have the same value for every class instance. Use instance attributes for properties that vary from one instance to another. instance attribute yazdırmak için çağırılmalıdır:

```
class Animal:
 def __init__(self):
 self.tür = "Köpek"
 print(self.tür)
Animal() >>> köpek
```

- `__init__` : this method aka initialization is similar to constructors in C++ and Java. Constructors are used to initializing the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated:

```
class DemoClass:
 # constructor
 def __init__(self):
 # initializing instance variable
 self.num= 100
def read_number(self):
 print(self.num)

creating object of the class. This invokes constructor
obj = DemoClass()

calling the instance method using the object obj
obj.read_number() >>> 100
```

an object cannot be created if we don't have a constructor in our program. This is why when we do not declare a constructor in our program, python does it for us.

there are 2 type of constructors:

1. default constructor: The default constructor is simple constructor which doesn't accept any arguments. It's definition has only one argument which is a reference to the instance which is being constructed.
2. parameterized constructor: constructor with parameters is known as parameterized constructor. The parameterized constructor take its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

1. default constructor:

```
def __init__(self):
 self.num = 999
```

2. parameterized constructor:

```
def __init__(self, length):
 self.num = 999
 self.length = 3
```

yani aralarındaki fark parametre alan / almayan.

`__init__(self)` 'ten sonra parametre eklemenin faydası çok daha az kod yazdırmasıdır:

```
class A:
 def __init__(self):
 self.isim = ' '
 self.yaş = 0
 self.memleket = ' '
p1 = A()
p1.isim = 'bugra'
p1.yaş = '19'
p1.memleket = 'artvin'
p1.append(A)
```

tek tek objeleri atamak yerine self'ten sonra parametre ekleyebiliriz:

```
class A:
 def __init__(self, isim, yaş, memleket):
 self.isim = isim
 self.yaş = yaş
 self.memleket = memleket
p1 = A('buğra',19,'artvin')
```

okumayı arttırmak için başlarına constructor'ın isimlerini de tanımlayabilirsin:

```
p1 = A(isim='buğra',yaş=19,memleket='artvin')
```

- in python, classes are objects. yani bir class'ı çağırdığın zaman python kullanabilmen için o class'ı bir objeye dönüştürür. When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances. an object consists of:
  - o identity: It gives a unique name to an object and enables one object to interact with other objects.
  - o state: it is represented by the attributes of an object. It also reflects the properties of an object.
  - o behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

objects are instances of a class:

```
harley = Bike()
```

the Bike is the class and harley is the object of this class. every object must be defined using the class. it's called instantiation.

```
class Bike:
 def brand(self):
 print("brand")
 def model(self):
 print("model")
harley = Bike()
```

class objects support two kinds of operations:

3. attribute references
4. instantiation (somutlaştırma)

to assing an item into that class named My\_class, you say:

```
class My_class:
 a = 5 # attribute reference
p1 = my_class() # instantiation
print(p1.a) >>> 5
```

örnek:

```
class Animal:
 tür = "köpek"
max = Animal()
print(max.tür) >>> köpek
```

max instance'ı tür olarak otomatik köpek değerini içerir çünkü Animal() sınıfına atadık.

- instantiate ettikten sonra çağırma:

```
class classİsmi():
 def __init__(self, birinciParametre, ikinciParametre):
 self.birinciParametre = birinciParametre
 self.ikinciParametre = ikinciParametre
nesneİsmi = classİsmi("aa","bb")
print(nesneİsmi.birinciParametre) >>> aa
print(nesneİsmi.ikinciParametre) >>> bb
```

- class ve instance attribute arasındaki farklardan biri şudur: class attribute'ları yazdırmak için çağırmanıza gerek yoktur. instance attribute'ları ise çağırmanız zorundasınız. mesela bu bir class attribute:

```
class Person:
 age = 15
 print(age) >>> 15
```

class attribute'lara atanan bir özellik o class'tan daha sonra oluşturulan bütün örnekler için otomatik atanır:

```
class Student():
 sınıf = 9
ahmet = Student()
mehmet = Student()
print(mehmet.sınıf) >>> 9
print(ahmet.sınıf) >>> 9
```

bu da instance attribute:

```
class Person:
 def __init__(self):
 self.age = 15
 print(self.age) >>> *yazdırmaz*
```

instance attribute'ları sonrasında mutlaka çağırmanız gerekir. Person() ile çağırılmazsa sadece print fonksiyonu yazdırmayacaktır.

yazdırması için çağırmanız gerekir:

```
class Person:
 def __init__(self):
 self.age = 15
 print(self.age)
Person() >>> 15
```

bir diğer fark da şudur: instance attribute'lara class attribute'ların aksine classİsmi.attribute diyerek erişemezsiniz. mesela class attribute'da:

```
class Person:
 age = 15
print(Person.age) >>> 15
```

yazdıracaktır ama

```
class Person:
 def __init__(self):
 self.age = 15
 print(self.age)
Person.age >>> *yazdırmaz*
```

yazdırması için classİsmi'nden sonra () koymanız gerekir:

```
Person().age >>> 15
```

- örnek:

```
class User:

 def __init__(self, userName, userSurname):
 self.follower = 0
 self.following = 0

 def follow(self, user):
 user.follower += 1
 self.following += 1
```

```
user1 = User()
user2 = User()
```



```
user1.follow(user2)

print(user1.follower) >>> 0
print(user1.following) >>> 1
print(user2.follower) >>> 1
print(user2.following) >>> 0
```

- class değil de instance attribute kullanmamızın sebeplerinden biri şudur: bir class attribute tanımladığınızda o attribute (mesela age = 15) o class'tan instantiate edilen bütün objectler için (ahmet(), mehmet()) geçerli oluyor.
- there are 3 kinds of methods (functions used in a class):
  1. class methods
  2. instance methods
  3. static methods

örnek nitelikleri üzerinde işlem yapacağımız zaman örnek metotlarını kullanıyoruz. Aynı şekilde sınıf nitelikleri üzerinde işlem yapacağımız zaman ise sınıf metotlarından faydalaniyoruz. Örnek metotları içinde herhangi bir örnek niteliğine erişmek istediğimizde self kelimesini kullanıyoruz. Sınıf metotları içinde bir sınıf niteliğine erişmek için ise cls kelimesini kullanıyoruz. İşte eğer bir sınıf içindeki herhangi bir fonksiyonda örnek veya sınıf niteliklerinin hiçbirine erişmeniz gerekmiyorsa, statik metotları kullanabilirsiniz.

1. class method:

```
class Math():

 @classmethod
 def pi(cls):
 return 3.14

 @classmethod
 def karekök(cls, sayı):
 return sayı ** 0.5
```

class methodları hem instancelar (object) üzerinden hem de class ismi üzerinden fonksiyonlara erişebiliyor:

```
print(Math.pi()) >>> 3.14
a = Math()
print(a.pi()) >>> 3.14
```

2. instance method:

```
class Math():

 def pi(self):
 return 3.14

 def karekök(self, sayı):
 return sayı ** 0.5

a = Math()

print(a.pi()) >>> 3.14
print(a.karekök(144)) >>> 12.0
```

bilindiği üzere instance methodları class ismi üzerinden çağırıyoruz, a isminde bir obje yaratıp onun üzerinden çağırdık.

3. static method: sınıf metotlarına çok benzer. Tıpkı sınıf metotlarında olduğu gibi, anlamsal olarak sınıfla ilgili olan, ancak sınıf metotlarının aksine bu sınıfın herhangi

bir niteliğine erişmesine gerek olmayan fonksiyonları, sınıf dışına atmak yerine, birer statik metot olarak sınıf içine yerleştirebiliriz.

örnek:

```
class Math():

 @staticmethod
 def pi():
 return 3.14

 @staticmethod
 def karekök(sayi):
 return sayi ** 0.5
```

gördüğün gibi iki fonksiyon da içine self ya da cls almadı çünkü bu iki sınıfın da sınıf veya örnek nitelikleriyle herhangi bir işi yok. bu kodu mat.py'ye kaydedip de kullanabiliriz:

```
from mat import Math
m = Math()
print(m.pi())
```

# ya da

```
print(m.karekök(144))
```

Üçünün bir arada gösterimi:

```
class Sınıf():

 # that's a class attribute
 sınıf_niteliği = 0

 # that's an instance attribute
 def __init__(self, veri):
 self.veri = veri

 # that's an instance method
 def örnek_metodu(self):
 return self.veri

 # that's a class method
 @classmethod
 def sınıf_metodu(cls):
 return cls.sınıf_niteliği

 # and that's a static method that you use if you don't
 # need to access neither class nor instance attributes
 @staticmethod
 def statik_metot():
 print('merhaba statik metot!')
```

- instance attributes precedes class attributes that is if you have two kind of attributes (class-instance) with the same value, instance attribute's value will be printed when you called them at the same time. if you want to access class attribute, you must call the instance with the className.attribute, not classInstance.attribute:

```
class Person:
 age = 20
 def __init__(self):
 self.age = 15
```

```
sınıf ismiyle instance'ı çağırırsak 20
print(Person.age) >>> 20
```

```
attribute ismiyle instance'ı çağırırsak 15
ahmet = Person()
print(ahmet.age) >>> 15
```

- ayrıca bir önemli husus da şudur: fonksiyon blogunun (def \_\_init\_\_()) haricinde instance attribute (self.attr = attr) yazamazsınız. mesela şöyle bir kod yanlıştır:

```
class A():
 self.age = 15
 def __init__(self):
 self.name = "name"
```

age satırını da def'in bloguna yazmalısın

- self bir keyword değildir yani başka herhangi bir kelime de olabilir (devamında kelime.age = age demek şartıyla). mühim olan initten sonra gelen ilk kelime self görevi görür. ama %99 self kullanılır.
- class sample with attribute:

```
class Orange():
 def __init__(self, color):
 self.color = color
orange = Orange("blue")
print(orange.color) >>> blue
```

\_\_init\_\_() blogunun altında self.attribute = attribute dememizin sebebi bu attribute'u class'ın başka yerlerinde de kullanabilmek içindir. yani attribute isimli parametreyi self.attribute kodu yardımıyla instance attribute haline getiriyoruz.

örnek:

```
class Animal():
 def __init__(self, name):
 self.name = name
animal = Animal("max")
print(animal.name) >>> max
```

örnek2:

```
class Animal():
 def __init__(self, name):
 self.name = name
animal = Animal("max")
print(Animal.__name__) >>> Animal
```

to change the value of an attribute, you simply:

```
class Orange():
 def __init__(self, color):
 self.color = color
orange = Orange("blue")
orange.color = "yellow"
print(orange.color) >>> yellow
```

- four pillars of OOP:
  1. Inheritance: child class (aka sub classes) possesses all the properties and behaviors of its parent class (aka base class, super class)

2. Polymorphism: the ability to present the same interface (functions, methods etc.) for differing underlying data types. mesela print ile hem True hem "abc" hem de 150 yazdırmak gibi.
3. Abstraction: bu özelliği bir class yaratıp metotlarını tanımladığımızda kullanıyoruz. Say we create a class to represent a person. When we define our person class- and the methods that go with it- we are creating an abstraction.
4. Encapsulation: In object-oriented programming you can restrict access to methods and variables - we can make the methods and variables private. This can prevent the data from being modied by accident and is known as encapsulation. it hides the implementation details of a class from other objects. yani In object-oriented programming, encapsulation refers to the bundling of data with the methods that operate on that data, or the restricting of direct access to some of an object's components.

#### 5. (+) Composition

- misal, bir köpeğin adı ve yaşı vardır, oturur ve uyurlar. bunları belirli bir köpeğe atfedebilmek için Dog isminde bir class oluşturalım:

```
class Dog(): #1
 def __init__(self, name, age): #2
 self.name = name #3
 self.age = age
 def sit(self): #4
 print(self.name.title() + " is now sitting.")
 def sleep(self):
 print(self.name.title() + "is now sleeping.")
```

1. class'ların ilk harfi adeten büyük yazılır and since we create this class from scratch, parantheses of class name is empty
2. The \_\_init\_\_() method is a special method Python runs automatically whenever we create a new instance based on the Dog class.
3. We'll pass Dog() a name and an age as arguments; self is passed automatically, so we don't need to pass it. Whenever we want to make an instance from the Dog class, we'll provide values for only the last two parameters, name and age. The two variables defined (name, age) have the prefix "self". Any variable prefixed with self is available to every method in the class, and we'll also be able to access these variables through any instance created from the class. Variables that are accessible through instances like this are called attributes.
4. The Dog class has two other methods defined: sit() and sleep(). Because these methods don't need additional information like a name or age, we just define them to have one parameter, self.

- örnek:

```
class Kişi:
 def __init__(self, name):
 self.name = name
 def say_hi(self):
 print(f"hello my name is {self.name}")
p = Kişi("Buğra")
p.say_hi() >>> hello my name is Buğra
```

1. kişi adında bir class tanımladık
2. daha sonra \_\_init\_\_(self, parametre) metoduyla bir name parametresi ekledik (parameterized constructor)
3. bunu self.name = name diyerek kendisine atadık, nesneleştirdik
4. daha sonra merhaba demeye yaran bir fonksiyon tanımladık ve altına print'li bir görev atadık

5. p isminde bir kişi tanımladık ve Kişi class'ına "Buğra" string'ini atarak bu classın name attribute'uyla eşleştirdik
6. finalde de object üzerinden (p) fonksiyonu (say\_hi()) çağırdık

örnek2:

```
class Dog:
 animal = 'dog'
 def __init__(self, age, color):
 self.age = age
 self.color = color
Max = Dog('6', 'black and white')
Tarçın = Dog('2', 'brown')
print("Max's details:")
print("Max's age is", Max.age)
print("Max's color is", Max.color)
print("Tarçın's details:")
print("Tarçın's age", Tarçın.age)
print("Tarçın's color", Tarçın.color) >>>
Max's details:
Max's age is 6
Max's color is black and white
Tarçın's details:
Tarçın's age 2
Tarçın's color brown
```

örnek3:

```
class Toplama():
 first = 0
 second = 0
 result = 0
 def __init__(self, a, b):
 self.first = a
 self.second = b
 def display(self):
 print("first number:", str(self.first))
 print("second number:", str(self.second))
 print("result is:", str(self.result))
 def calculate(self):
 self.result = self.first + self.second
obj = Toplama(10, 20)
obj.display() >>>
first number: 10
second number: 20
result is: 0
```

- her class tanımlamasından sonra docstring içinde ("""" , #) ne işe yaradığı yazılır ki hatırlanması kolay olsun.
- destructors are called when an object gets destroyed. In Python, destructors are not needed as much needed in C++ because Python has a garbage collector that handles memory management automatically. The \_\_del\_\_() method is known as a destructor method in Python. It is called when all references to the object have been deleted. Şeması şu şekildedir:

```
def __del__(self):
 # body of destructor
```

\_\_del\_\_() örnek:

```
class Employee:
 def __init__(self):
```

```

 print("employee created.")
 def __del__(self):
 print("destructor called, employee deleted.")
obj = Employee()
del obj

```

- `__main__` is an identifier used to reference the current file context. When a module is read from standard input, a script, or from an interactive prompt, its `__name__` is set equal to `__main__`.
- `dir()` metoduyla bir classın muhtemel metotlarını görürsün:

```

class Cemo:
 pass
print(dir(Cemo)) >>> ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__',
 '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
 '__weakref__']

```

these methods are called dunder methods (because of double underscore `__`)

- Methods are functions defined inside the body of a class. They are used to perform operations with the attributes of our objects. Methods are essential in the encapsulation concept of the OOP paradigm. In Python, we can call methods in two ways:
  1. bounded methods
  2. unbounded methods

```

class Methods:
 def __init__(self):
 self.name = 'Methods'
 def getName(self):
 return self.name
m = Methods()
print(m.getName()) #1
print(Methods.getName(m)) #2 >>> ikisi de Methods yazdırır.

```

1. this is the **bounded** method call. The Python interpreter automatically pairs the `m` instance with the `self` parameter.
  2. and this is the **unbounded** method call. The instance object is explicitly given to the `getName()` method.
- örnek:

```

class Person():
 def __init__(self, name, year):
 self.name = name
 self.year = year
 def intro(self):
 print("senin adın " + self.name + " ve yaşın " + str(self.year))
 def calculateAge(self):
 return 2021 - self.year
p1 = Person(name = "bugra", year= 2001)
p2 = Person(name = "cemre", year= 2011)
print(f"adım: {p1.name} ve yaşım {p1.calculateAge()}")
print(f"adım: {p2.name} ve yaşım {p2.calculateAge()}") >>>
adım: bugra ve yaşım 20
adım: cemre ve yaşım 10

```

- daire hesaplama:

```

class Circle:
 pi = 3.14
 def __init__(self, yarıcap=0):
 self.yarıcap = yarıcap

```

```

def cevre_hesapla(self):
 return 2*self.pi*self.yaricap
def alan_hesapla(self):
 return self.pi * (self.yaricap ** 2)
daire1 = Circle(yaricap=8)
daire2 = Circle(yaricap=3)
print(f"daire1'in cevresi {daire1.cevre_hesapla()} alanı ise {daire1.alan_hesapla()}")
print(f"daire2'in cevresi {daire2.cevre_hesapla()} alanı ise {daire2.alan_hesapla()}")

```

- `__str__()` methodu ile bir class'ın ismini değiştirebilirsin:

```

class Dog:
 def __init__(self, age, name):
 self.name = name
 self.age = age
a = Dog(5, "max")
print(a) >>> <__main__.Dog object at 0x0000001EF88D9A760>

```

bu uzun kriptonik adresi vermesin istiyorsan:

```

class Dog:
 def __init__(self, age, name):
 self.name = name
 self.age = age
 def __str__(self):
 return f"{self.name} is {self.age} years old."
a = Dog(5, "max")
print(a) >>> max is 5 years old.

```

- örnek:

```

class Resturant():
 def __init__(self, name, cuisine_type):
 self.name = name
 self.cuisine_type = cuisine_type
 def describe_resturant(self):
 result = f"{self.name} restoranı harika {self.cuisine_type} yapar."
 print(f"{result}")
 def open_resturant(self):
 open = f"{self.name} şuan açıktır."
 print(f"{open}")
restaurant = Resturant("Little Cesar's", "pizza")
restaurant.open_resturant()
restaurant.describe_resturant()
restaurant2 = Resturant("burger town", "burger")
restaurant2.open_resturant()
restaurant2.describe_resturant()
>>>
Little Cesar's şuan açıktır.
Little Cesar's restoranı harika pizza yapar.
burger town şuan açıktır.
burger town restoranı harika burger yapar.

```

örnek2:

```

class User():
 def __init__(self, first_name, last_name, nickname, password, age):
 self.first_name = first_name
 self.last_name = last_name
 self.nickname = nickname
 self.password = password
 self.age = age

```

```

def describe_user(self):
 print(f"{self.first_name} {self.last_name}")
 print(f"nickname: {self.nickname}")
 print(f"password: {self.password}")
 print(f"age: {self.age}")

def greet_user(self):
 print(f"Welcome back {self.nickname} ! ")

eskoda = User("Bugra", "Kara", "eskoda", "bugra123", 19)
eskoda.describe_user()
eskoda.greet_user() >>>
Bugra Kara
nickname: eskoda
password: bugra123
age: 19
Welcome back eskoda !

```

örnek3:

```

class Car():
 def __init__(self, make, model, year):
 self.make = make
 self.model = model
 self.year = year

 def name(self):
 long_name = str(self.year) + " " + self.make + " " + self.model
 return long_name.upper()

my_car = Car("fiat", "fiorino", 2018)
print(my_car.name()) >>> 2018 FIAT FIORINO

```

bunu yukarıdakinden farklı yazdığımız için en altta fonksiyonu da printli yazmak zorunda kaldık, direkt çağırıyoruz. yukarıdaki gibi yapmak için printi def'in blogunda yazman gerek:

```

class Car():
 def __init__(self, make, model, year):
 self.make = make
 self.model = model
 self.year = year

 def name(self):
 print(f" year is: {self.year} make is {self.make} and model is {self.model}")

my_car = Car(2018, "fiat", "fiorino")
my_car.name()

```

- we can set a default value for an attribute:

```

class Game():
 def __init__(self, name, length):
 self.name = name
 self.length = length
 self.out_date = 2015

 def game_description(self):
 print(f"the name of the game is {self.name} and its length is {self.length}")

 def date(self):
 print("the out date is " + str(self.out_date) + ".")

oyun = Game("Crysis", "20+ hours")

```



```
oyun.game_description()
oyun.date() >>>
the name of the game is Crysis and its length is 20+ hours
the out date is 2015.
```

- you can change an attribute's value in three ways:
  1. change the value directly through an instance
  2. set the value through a method
  3. increment the value (add a certain amount to it) through a method
  1. the easiest way to modify the value of an attribute is to access the attribute directly through an instance.

yukarıdaki örnekte out\_date'i 2013 olarak değiştirmek için:

```
oyun.out_date = 2013
oyun.out_date() >>> the out date is 2015.
```

2. another way is to update an attribute's value through a method. Instead of accessing the attribute directly, you pass the new value to a method that handles the updating internally:

3. another way is to increment value through a method.

- inheritance metodu: class yazarken her seferinde en baştan yazmak yerine, eğer yazacağın class daha önceden yazdığının bir alt dalıysa, inheritance metodunu kullanabilirsin. When one class inherits from another, it automatically takes on all the attributes and methods of the first class. the original class is called parent class and the new class is child class. Child class takes every method and attribute of its parent class but is also free to create new attributes and methods. The first task Python has when creating an instance from a child class is to assign values to all attributes in the parent class. To do this, the `_init_()` method for a child class needs help from its parent class. A child class needs to identify which class is its parent class. This can be done by mentioning the parent class name in the definition of the child class:

```
class ChildClass(ParentClass):
```

child class overrides or extends the attributes and behaviors of its parent class. child class must always come after the parent class in code line.

örnek:

```
class Artist():

 def __init__(self, name, country):
 self.name = name
 self.country = country

 def intro(self):
 print(f"this painter's name is {self.name} and he is from {self.country}.")

class SalvadorDali(Artist):
 o1 = Artist("salvador dali","spain")

 o1.intro() >>> this painter's name is salvador dali and he is from spain.
```

notice that you write your objects indented under the child class

- örnek:

```
class Animal:

 def __init__(self,name,age):
 self.name = name
 self.age = age

class Dog(Animal):
```

```
def sound(self):
 print("Woof!")
```

```
Max = Dog("Max",4)
print(Max.name)
print(Max.age) >>> Max 4
```

- The `__init__()` function of a childClass overrides the `__init__()` function of its parent class:

```
class childClass(parentClass):
 def __init__(self, name, surname):
```

When you add the `__init__()` function, the child class will no longer inherit the parent's `__init__()` function. because the child's `__init__()` function overrides the inheritance of the parent's `__init__()` function.

eğer child class parent class'In `_init_()` argümanlarını tutsun istiyorsan you should add a call to parent's `_init_()`:

```
class childClass(parentClass):
 def __init__(self, name, surname):
 parentClass.__init__(self, name, surname)
```

- inheritance türleri:

1. multiple inheritance
2. hierarchial inheritance
3. hybrid inheritance

1. multiple inheritance şu demektir, When a class inherits the method and attributes from multiple parent class then it is called multiple inheritance. This allows us to use the property from multiple base classes or parent classes in a derived or child class. Şeması şöyledir:

```
class childClass(parent1, parent2, parent3):
```

örnek:

```
class Mother:
 mothername = ""
 def mother(self):
 print(self.mothername)

class Father:
 fathername = ""
 def father(self):
 print(self.fathername)

class Son(Mother, Father):
 def parents(self):
 print("Father:",self.fathername)
 print("Mother:",self.mothername)
```

```
s1 = Son()
s1.fathername = "Zeki"
s1.mothername = "Tülay"
s1.parents() >>>
Father: Zeki
Mother: Tülay
```

aynı metot parent classlarda mevcutsa çoklu miras alırken ilk önce hangisinin adını yazıyorsan o class'taki değerini alır.

2. hierarchial inheritance: When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In this program, we have a parent (base) class and two child (derived) classes.

örnek:

```
class Parent:
 def func1(self):
 print("this function is in parent class.")

class Child1(Parent):
 def func2(self):
 print("this function is in child1.")

class Child2(Parent):
 def func3(self):
 print("this function is in child 2.")

object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3() >>>
this function is in parent class.
this function is in child1.
this function is in parent class.
this function is in child 2.
```

3. hybrid inheritance: Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

örnek:

```
class School:
 def func1(self):
 print("This function is in school.")

class Student1(School):
 def func2(self):
 print("This function is in student 1. ")

class Student2(School):
 def func3(self):
 print("This function is in student 2.")

class Student3(Student1, School):
 def func4(self):
 print("This function is in student 3.")

object = Student3()
object.func1()
object.func2()
>>>
This function is in school.
This function is in student 1.
```

- In inheritance, the child class inherits the methods from the parent class. However, it is possible to modify a method in a child class that it has inherited from the parent class. This is particularly useful in cases where the method inherited from the parent class doesn't quite fit the child class. In such cases, we re-implement the method in the child class. This process of re-implementing a method in the child class is known as Method Overriding.
- miras alma mekanizmasına göre 3 çeşit yol vardır:

1. Miras alınan sınıfın bütün nitelik ve metotları alt sınıfa olduğu gibi devredilir
2. Miras alınan sınıfın bazı nitelik ve metotları alt sınıfta yeniden tanımlanır
3. Miras alınan sınıfın bazı nitelik ve metotları alt sınıfta değişikliğe uğratılır

1. eğer child class'ın altında bir değişiklik yapmazsan aynen devredilir:

```
class Oyuncu():

 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 0

class Asker(Oyuncu):
 pass
```

mesela bu 1. tür miras almaya örnektir çünkü herhangi bir reevalute, override vs. yapmadık. bir kopyası gibidir. hatta eğer child class'ta herhangi bir değişiklik yapmayacak, yeni bir bilgi eklemeyecekseniz direkt parent class üzerinden iş yapmak daha mantıklıdır.

2. parent class'ın özelliklerini devralmanın yanısıra, you may add new informations in the child class:

```
class Oyuncu():

 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 0

class Asker(Oyuncu):
 can = 100

class İşçi(Oyuncu):
 pass
```

can = 100 bilgisi sadece Asker child class'ına özgüdür. eğer can = 20 diye bir kod parent class'ta (Oyuncu) olsaydı child class'ta (Asker) yeniden yazılan can = 100 bilgisi bu kodu override ederdi. fakat aynı kodu ayrı ayrı child class'larda farklı değerlerle yazıp instantiation esnasında atadığın nesne isimleriyle çağırırsan sorun olmayacaktır:

```
class Asker(Oyuncu):
 can = 130

class İşçi(Oyuncu):
 can = 55

nesne1 = Asker("bugra", "kara")
nesne2 = İşçi("cemre", "kara")

print(nesne1.can) >>> 130
print(nesne2.can) >>> 55
```

hepsinin değerini \_\_init\_\_()’le atamak istersek şöyle yazmak zorundayız:

```
class Oyuncu():

 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 0
```

```
class Asker(Oyuncu):
 def __init__(self, isim, rütbe):
 self.güç = 70

class İşçi(Oyuncu):
 def __init__(self, isim, rütbe):
 self.güç = 30
```

fakat init koduna yazdığımız isim ve rütbe değerlerini init blogunda self.isim = isim ya da self.rütbe = rütbe demedik. child classlarda belirttiğin \_\_init\_\_() satırları parent classinkileri elimine ediyor. dolayısıyla çalıştırmak istesek hata verecektir:

```
nesne = asd.Asker("bugra","kara")
nesne.isim >>> AttributeError: 'Asker' object has no attribute 'isim'
```

bu sorunu çözmek için child classları şöyle yazmamız gerekirdi:

```
class Oyuncu():

 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 0

class Asker(Oyuncu):
 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 70

class İşçi(Oyuncu):
 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 30
```

```
nesne = asd.Asker("bugra","kara")
print(nesne.isim)
```

fakat bu sefer de dünya kadar kod yazmak zorunda kaldık ki bu da miras kavramına aykırı. miras etme sebebimiz zaten mükerrer kod yazmayalım diyeydi.

- işte bunun için nihai bir metot var: super() metodu. şöyle yazılır:

```
super().__init__(param1, param2)
```

super() function makes the child class inherit all the methods and properties from its parent class.

```
class childClass(parentClass):
 def __init__(self, name, surname):
 super().__init__(name, surname)
```

By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

asker kodunun super() ile yazılmış çok daha kısa hali:

```
class Oyuncu():

 def __init__(self, isim, rütbe):
 self.isim = isim
 self.rütbe = rütbe
 self.güç = 0

class Asker(Oyuncu):
```

```
def __init__(self, isim, rütbe):
 super().__init__(isim, rütbe)
 self.güç = 70
```

bu satırdaki super() fonksiyonu, tam da adının anlamına uygun olarak, miras alınan üst sınıfın \_\_init\_\_() metodu içindeki kodların, miras alan alt sınıfın \_\_init\_\_() metodu içine aktarılmasını sağlıyor. Böylece hem taban sınıfın \_\_init\_\_() metodu içindeki self.isim ve self.rütbe niteliklerini korumuş, hem de self.güç adlı yeni bir nitelik ekleme imkanı elde etmiş oluyoruz.

```
class Asker(Öyuncu):
 def __init__(self, isim, rütbe):
 super().__init__(isim, rütbe)
 self.güç = 70
```

gördüğünüz gibi, super() fonksiyonu sayesinde taban sınıfın değiştirmek istediğimiz niteliklerine yeni değerler atarken, değiştirmek istemediğimiz nitelikleri ise aynı şekilde muhafaza ettik.

eğer init satırında parametreleri tek tek yazmak istemiyosan \*args metodunu kullanabilirsin:

```
class Asker(Öyuncu):
 def __init__(self, *args):
 super().__init__(*args)
 self.güç = 30
```

to add a variable to a child class with super() method, you just add with self.attr = x :

```
class Student(Person):
 def __init__(self, fname, lname):
 super().__init__(fname, lname)
 self.graduationyear = 2019
```

super() metodunu sadece initle değil metotlarla da kullanabilirsin:

```
class parentClass():
 def __init__(self, p1, p2):
 self.p1 = p1
 self.p2 = p2
 self.myValue = 1000

 def kodYaz(self):
 print("kod yazılıyor...")

class childClass(parentClass):
 def __init__(self, p1, p2):
 super().__init__(p1, p2)
 self.myValue = 1500

 def kodYaz(self):
 super().kodYaz()
 print("tebrikler, kod yazıldı")
```

```
nesne = childClass("a", "b")
print(nesne.kodYaz()) >>>
kod yazılıyor...
tebrikler, kod yazıldı
```

gördüğün gibi sadece init değil metotlarda da değişiklik yapabiliyoruz super() metodu ile. parentClass'ın kodYaz fonksiyonuna yeni bir print işlemi ekledik.

- You can modify properties on objects like this:

```
o1.age = 40
```

You can delete properties on objects by using the del keyword:

```
del o1.age
```

or you can delete the whole object:

```
del o1
```

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error:

```
class Person:
 pass
```

- integer, value, string, function, tuples vs. bir class'ın nesneleri yani üyesi olabilirler. 3 çeşit class üyesi vardır:

1. gizli
2. açık
3. yarı-gizli

1. gizli: bir üyeyi gizli yapmak için başına dunder ( \_ \_ ) koyarız. sonuna en fazla bir tane olmak üzere koyabiliriz:

```
class Deneme:
 __gizli = "gizli üye"
```

2. açık:

3. yarı gizli: bir üyeyi yarı gizli yapmak için başına score ( \_ ) koyarız

- python'da bir class'ın 3 aşaması vardır:

- o construction (inşa) (class ...)
- o initialization (ilklendirme) (def \_\_init\_\_(self))
- o destruction (sonlandırma) (del ...)

1. class diyerek sınıfı yaratıyoruz

2. Python'da bir sınıfı ilklendirmek için \_\_init\_\_() adlı bir metottan yararlanıyoruz.

Ancak, adının aksine, ilklendirme, sınıfların oluşturulmasına ilişkin ilk basamak değildir. Python, bir sınıfın ilklendirilmesinden önce o sınıfı inşa eder. Bu inşa işleminden sorumlu metodun adı ise \_\_new\_\_() 'dur. bilindiği gibi pythonda basit bir class şöyle tanımlanır:

```
asd.py
```

```
class Sınıf:
 def __init__(self):
 print("merhaba sınıf")
```

burada \_\_init\_\_() metodu, sınıfımız örneklenir örneklenmez hangi işlemlerin yapılacağını gösteriyor. Yani mesela sınıf = Sınıf() gibi bir kod yardımıyla Sınıf() adlı sınıfı örneklediğimiz anda ne olacağını bu \_\_init\_\_() metodu içinde tanımlıyoruz:

```
import asd
snf = asd.Sınıf() >>> merhaba sınıf
```

Gördüğümüz gibi, tam da \_\_init\_\_() metodunda tanımladığımız şekilde, sınıfımızı örneklediğimiz anda ekrana 'merhaba sınıf' çıktısı verildi. Ancak yukarıda da belirttiğimiz gibi, bir sınıf örneklendiğinde çalışan ilk metod aslında \_\_init\_\_() değildir. Python bu süreçte alttan alta \_\_new\_\_() adlı başka bir metodu çalıştırır. eğer bunun üzerinde değişiklik yaparsanız (asd.py'de):

```
class Sınıf:
 def __new__(cls):
 pass

 def __init__(self):
 print("merhaba sınıf")
```

artık snf kodu çalışmayacaktır. Eğer \_\_new\_\_() metodunun öntanımlı davranışını taklit etmek isterseniz yukarıdaki kodları şu şekilde yazmalısınız:

```
class Sınıf():
 def __new__(cls, *args, **kwargs):
 return object.__new__(cls, *args, **kwargs)
```

```
def __init__(self):
 print('merhaba sınıf')
```

çünkü normalde de \_\_new\_\_() fonksiyonunun içeriği böyledir.

3. del classismi diyerek de o classı siliyoruz



## #modules

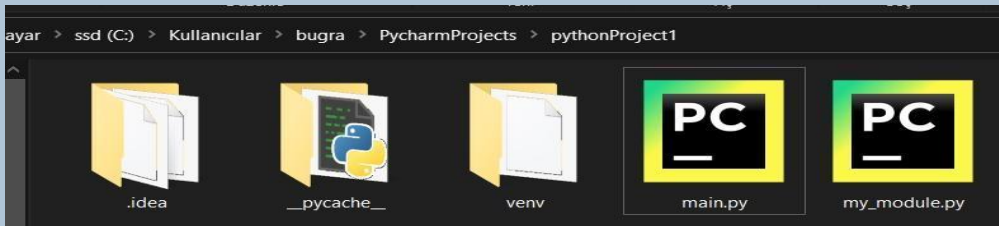
- A module is a file containing Python code. Python code can be managed using:
  1. functions
  2. classes
  3. modules
  4. packages

Python modules are used to organize Python code, they are like libraries storing information at hand and can be called and used when needed. For example, mathematical related built-in functions are placed inside the math module, functions about randomness in the math module etc. Modules are grouped together to form packages. 2 çeşit modül vardır:

1. kendi hazırladığımız modüller
  2. built-in modüller
    - standart kütüphane modülleri
    - 3. şahıs modülleri (açık kaynak) havuzu. bu havuzun adı pypi.org (pip install x).
- To create a module just save the code you want in a file with the file extension .py. misal aşağıya bir kod yazalım:

```
my_module.py
def greeting(name):
 print("Hello, " + name)
```

bu over-simplified kodu my\_module.py ismiyle Save As diyerek aynı dizine kaydedelim.



daha sonra herhangi bir PyCharm sekmesinde bu dosyayı module ederek bu fonksiyonu tekrar yazmadan, sadece istediğimiz veriyi girerek ekrana yazdırabiliriz:

```
import my_module
my_module.greeting("buğra") >>> Hello, buğra
```

- bir modül sadece function değil variable, array, dicts, objects vs. de içerebilir. misal bir başka örnek:

```
person1 = {
 "name": "John",
 "age": 36,
 "country": "Norway"
}
```

bu kodu modül2.py ismiyle kaydedelim. daha sonra importla çağırıp age'in value'sunu soralım:

```
import modül2
a = modül2.person1["age"]
print(a) >>> 36
```

- datetime modülü ile zaman:

```
import datetime
x = datetime.datetime.now()
print(x)
```

istersen .now()'dan sonra month year day hour minute vs. sorabilirsin ctime metoduyla bu bilgiyi yazıyla alabilirsin:

```
import datetime
a = datetime.datetime.now().ctime()
print(a) >>> Mon Jul 5 15:43:01 2021
```

to create a date, we can use the datetime() class (constructor) of the datetime module. The datetime() class requires three parameters to create a date: year, month, day:

```
import datetime
x = datetime.datetime(2001, 7, 8, 23, 14, 25)
print(x) >>> 2001-07-08 23:14:25
```

strftime() metodu is used for formatting date objects into readable strings. for instance girilen ayı yazdırmak için:

```
import datetime
x = datetime.datetime(2018, 6, 1)
print(x.strftime("%B")) >>> June
```

6. ay June olduğu için June yazdırdı.

bazı kısayollar:

1. %a - weekday, short version - Wed
2. %A - weekday, long version - Wednesday
3. %w - weekday, as a number between 0-6 (0 is Sunday) - 3
4. %d - day of month between 01-31 - 25
5. %b - month name, short version - Dec
6. %B - month name, full version - December
7. %m - month as a number between 0-12 - 12
8. %Y - year
9. %H - hour between 00-23
10. %p - am/pm - pm
11. %M - minute between 00-59 - 30
12. %S - second between 00-59
13. %f - micro second between 000000-999999
14. %Z - timezone - CST
15. %c - local version of date and time - Mon Dec 31 17:41:00 2021
16. %x - local version of date - 12/31/2021
17. %X - local version of time - 17:30:25

- os modülü: genel olarak işletim sistemiyle ilgili bilgi veren bir modüldür. os modülüyle işletim sistemini öğrenmek için:

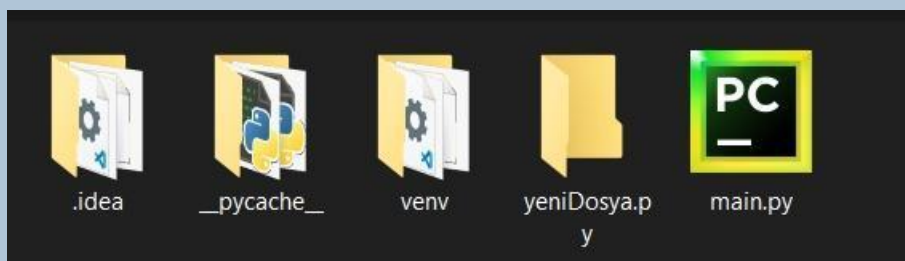
```
import os
a = os.name
print(a) >>> nt (nt windows demektir)
```

os.getcwd() metodu ile yazdığın dosyanın hangi path'de olduğunu öğrenebilirsin:

```
import os
a = os.getcwd()
print(a) >>> C:\Users\bugra\PycharmProjects\eskoda.py
```

os.mkdir() metodunu bu path'de yeni bir klasör (dosya değil) oluşturmak istiyorsan kullanmalısın:

```
import os
a = os.getcwd()
os.mkdir("yeniDosya.py")
```



os.chdir() metodunu eğer başka bir konumda oluşturmak istiyorsan kullanarak peşine konumu yazmalısın:

```
import os
os.chdir("C:\\users\\bugra\\Desktop")
os.mkdir("artvin") >>> masaüstünde artvin isimli bir klasör oluşturu
```

os.chdir() ile default klasörü değiştiriyoruz. o kodu okuyunca python odağını o dizine değiştiriyor. daha sonra yeni bir klasör oluştur diyince nereye diye sormadan otomatik olarak orada oluşturuyor. string içine " .. " koyarsan bir üst klasöre geçer:

```
os.chdir("../")
```

- iç içe klasörler oluşturmak için makedirs() metodu kullanmalısın:

```
import os
os.makedirs("deneme klasör/yeni klasör/son klasör")
```

- dizin listeleme, yani bir klasörün altındaki bütün klasörleri listelemek için listdir() metodu kullanmalısın:

```
import os
a = os.listdir("K:\\")
print(a) >>>
['#watched series', '$RECYCLE.BIN', 'aa', 'artist', 'bookmarks.html', 'boun', 'burs belgeleri', 'crypto', 'dosyalar', 'ebook', 'Fast Reading', 'huawei 5temmuz21', 'Languages', 'lightshot', 'msdownld.tmp', 'National Geographic Maps', 'PROGRAMLAR', 'python', 'Sesli Medya', 'spotify', 'System Volume Information', 'The Teaching Company~Burnedheal~', 'tribute', 'UMUT', 'wallpaper', 'wordler', '[Aijin] Jujutsu Kaisen (1st Season) [1080p][HEVC x265 10bit]', '[Sick-Fansubs] Berserk 2016 [720p]', 'çeviri', 'üni']
```

bunu loop ve if condition'la da kullanabilirsin. mesela bir klasördeki sadece .py uzantılıları görmek için:

```
import os
for i in os.listdir("K:\\"):
 if i.endswith(".py"):
 print(i)
```

- rename() metodunu bir klasörü yeniden adlandırmak için kullanmalısın:

```
import os
os.rename("yeniKlasör", "aaa") >>> yeniKlasör'ün adını aaa yapar.
```

- rmdir() metodunu ise klasör silmek için kullanırız:

```
import os
os.rmdir("aaa")
```

- removedirs() metoduyla / ile ayırdığımız klasörün içindeki bir klasörü silebiliriz:

```
import os
os.removedirs("yeniKlasör/denemeKlasörü")
```

- path() modülüyle de isim ve uzantı değişikliği yapabiliriz. mesela herhangi bir dosyanın konumunu öğrenmek için os.path.abspath() modülünü kullanırız:

```
import os
a = os.path.abspath("abc.py")
print(a) >>> C:\Users\bugra\PycharmProjects\abc.py\abc.py
```

- One advantage of functions is the way they separate blocks of code from your main program. By using descriptive names for your functions, your main program will be much easier to follow. You can go a step further by storing your functions in a separate file called a module and then importing that module into your main program. An import statement tells Python to make the code in a module available in the currently running program file. Storing your functions in a separate file allows you to hide the details of your program's code and focus on its higher-level logic. It also allows you to reuse functions in many different programs. When you store your functions in separate files, you can share those files with other programmers without having to share your entire program. Knowing how to import functions also allows you to use libraries of functions that other programmers have written. There are a few ways to do it:

#### 1. Importing an Entire Module

2. Importing Specific Functions
3. Using as to Give a Function an Alias
4. Using as to Give a Module an Alias
5. Importing All Functions in a Module

1 - to import an entire module:

```
import moduleName
```

mesela matematik.py modülü sayıları çarpan çarpma isimli bir fonksiyon içersin:

```
def çarpma(a, b):
 return a * b
```

daha sonra başka bir dosyada bu dosyayı import ederek bir kod yazalım:

```
import matematik
obj1 = matematik.çarpma(5, 6)
print(obj1) >>> 30
```

2 - to import only spesifik functions: bir modüldeki yalnızca belli isimli fonksiyonları alma misal matematik.py isimli bir python dosyasında toplama() isminde bir fonksiyon tanımladın:

```
def toplama (a,b):
 return a + b
```

matematik.py dosyasından sadece bu fonksiyonu kullanmak istiyorsan:

```
from matematik import toplama
obj1 = matematik.toplama(5, 8)
print(obj1) >>> 13
```

You can import as many functions as you want from a module by separating each function's name with a comma:

```
from matematik import toplama, çarpma
```

there are two ways to import with this method:

1. `from matematik import toplama`
2. `import matematik.toplama`

you can import classes as well.

3 - to give the function an alias: fonksiyona geçici bir isim verir. şeması şöyledir:

```
from matematik import toplama as t
```

bu modül tekniğini 2 farklı sebeple kullanabilirsin:

1. eğer import edeceğin fonksiyonun ismini başka bir yerde de kullanıyorsan
2. eğer fonksiyon ismi çok uzunsa

4 - to give the whole module an alias: modüle geçici bir isim verir. şeması şöyledir:

```
import matematik as m
```

then you may use it with other techniques:

```
from m import t
```

5 - bir module'deki bütün function'ları import etmek için ' \* ' kullanırız:

```
from araba import *
```

The asterisk in the import statement tells Python to copy every function from the module pizza into this program file. Because every function is imported, you can call each function by name without using the dot notation. However, it's best not to use this approach when you're working with larger modules that you didn't write: if the module has a function name that matches an existing name in your project, you can get some unexpected results. The best approach is to import the function or functions you want, or import the entire module and use the dot notation.

bu durumda fonksiyonların başına `math.` getirmek zorunda değilsin:

```
from math import *
a = factorial(5)
print(a)
```

öncesinde şöyle yazmak zorundaydın:

```
import math
a = math.factorial(5)
print(a)
```

- random da bir modüldür. random modülüyle isim verme metodu:

```
import random
male = bool(random.randint(0, 1))
if male:
 print("We will name him ahmet")
else:
 print("We will name her ayse") >>> %50 oranında ahmet ya da ayse verir.
```

örnek:

```
import random
while True:
 val = random.randint(1, 10)
 print(val, end=" ")
 if val == 6:
 break
print()
```

1 ile 10 arasındaki sayıları sayı 6 çıkana kadar random şekilde yazdırır  
istediğin iki aralıkta random float yazdırma uniform() metoduyla olur:

```
import random
a = random.uniform(3, 18)
print(a)
>>> 3 ile 18 arasında bir float yazdırır
```

- istediğin iki sayı arasında integer yazdırma randint() metoduyla olur, 3 ve 18 dahildir:

```
import random
a = random.randint(3, 18)
print(a)
```

max (in here 18) dahil olmasın istiyorsan randrange() metodunu kullanmalısın

```
import random
random.randrange(3, 18)
print(a) >>> 3, 4, ... , 17 arasından random yazdırır
```

randint(a, b) metodunda dikkat edilmesi gereken nokta a b'den her zaman küçük olmalıdır.  
aksi taktirde hata verir:

- sample() metoduyla liste içerisinde q adet random değeri yazdırır:

```
import random
sayilar = range(1,100)
a = random.sample(sayilar, 3)
print(a) >>> 1 ile 100 arasında random 3 sayıyı yazdırır
```

- shuffle() bir listedeki elemanları random olarak yeniden sıralar:

```
import random
sayilar = [1, 2, 3, 4, 5, 6] #normalde print(sayilar) deseydik aynı bu yazdırılacaktı
random.shuffle(sayilar)
print(sayilar) >>> [5, 2, 4, 6, 1, 3]
```

- choice() bir listeden random bir eleman seçer:

```
import random
sayilar = [1, 2, 3, 4, 5, 6]
a = random.choice(sayilar)
print(a) >>> 6
```

choices() bir listeden random bir eleman seçer ve liste halinde verir. choice() ile farkı budur:

```
import random
liste = [1, 2, 3, 4, 5, 6]
```

```
a = random.choices(liste)
```

```
print(a) >>> [6]
```

böyle yazdığında tek bir eleman seçer. listede birden çok olsun istiyorsan k= metodunu kullanmalısın:

```
import random
```

```
liste = [True, "buğra", 128, 821, False]
```

```
a = random.choices(liste, k=3)
```

```
print(a) >>> [821, False, False]
```

- 0-100 arası random bir integer:

```
import random
```

```
print(random.randrange(0,100))
```

- 0'la 1 arasında random bir float yazdırma, random.random() metoduyla olur:

```
import random
```

```
a = random.random()
```

```
print(a)
```

```
>>> dersin her çalıştırdığında yeni bir float sayı generate eder. 0.7274353675996468
```

ya da yukarıdakinin aynısını:

```
from random import random
```

```
print(random())
```

- bir listeden random eleman alma:

```
import random
```

```
araba = ["porsche", "suv", "honda", "gmc"]
```

```
result = araba[random.randint(0,len(araba)-1)]
```

```
print(result)
```

1. import'la random modülünü çağırdık
2. araba isminde bir liste tanımladık
3. random.randint() metoduyla 0'dan (0. indeksten) listenin sonuna kadar (sonuncu eleman dahil) bir aralık belirleyip rastgele bu aralıktan bir index numarası (integer) belirleyip bu indeks numarasıyla araba[ ] listesinden eleman atadık result değişkenine
4. daha sonra printle yazdırdık

burda önemli olan (0, 4) gibi bir kesinlik koymamamız, dolayısıyla listeye yeni elemanlar eklense de kod hata vermeyecektir. dinamik verilerle çalıştığında böyle yapmalısın.

- kullandığın işletim sistemini öğrenmek için bir built-in module:

```
import platform
```

```
x = platform.system()
```

```
print(x) >>> Windows
```

- help() metodunu modüllerin tek tek ne işe yaradıklarını öğrenmek istiyorsan kullanabilirsin:

```
import math
```

```
a = help(math)
```

```
print(a)
```

- import etmediğin fonksiyonu yazarsan hata verecektir. ayrıca import ettiğin modül ile çalıştığın dosyanın isminin aynı olmamasına dikkat et.
- pyinstaller modülü .py uzantılı dosyaları .exe'ye çevirmek için kullanılır. python dosyasının konumuna gidip konum satırına tıklayarak cmd yaz. daha sonra şu kodu gir:

```
pyinstaller --onefile -w dosyaİsmi.py
```

bu kod o dosyada dist isimli klasörde programı yaratacaktır. resim eklemek istiyorsan:

```
pyinstaller --onefile --icon="asd.png" -w dosyaİsmi.py
```

- pip'le eskimiş paketleri görmek için:

```
pip list --outdated
```

güncellemek için:

```
pip install [package_name] --upgrade
```

pip'in kendisini güncellemek için:

```
py -m pip install -U pip
```

- pyperclip modülü clipboard'daki kopyaladığın öğelere erişerek kolaylık sağlar:
  - o asd
-

## #RegularExpression

- A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern. to work with regular expressions, import re:

```
import re
```

Metacharacters are characters with a special meaning:

- o [ ] - a set of characters
- o \ - Signals a special sequence (can also be used to escape special characters)
- o ^ - starts with
- o \$ - ends with
- o \* - zero or more occurrences
- o + - one or more occurrences
- o { } - Exactly the specified number of occurrences
- o | - either or
- o \A - Returns a match if the specified characters are at the beginning of the string
- o \b - Returns a match where the specified characters are at the beginning or at the end of a word

some re functions:

1. findall - Returns a list containing all matches
2. search - Returns a Match object if there is a match anywhere in the string
3. split - Returns a list where the string has been split at each match
4. sub - Replaces one or many matches with a string

1. findall() function returns a list containing all matches.:

```
import re
txt = "127647236458716354815236715248735498236498623958623"
a = re.findall("2",txt)
print(len(a)) >>> 7
```

if there is no "2" it returns an empty list

findall metodunda [ ] içinde harf arayabilirsiniz:

```
import re
txt = "The rain in Spain"
a = re.findall("[ra]", txt)
print(a) >>> ['r', 'a', 'a']
```

r ve a harflerini arar buldukça listeye ekler. mesela [a-e] aratırsan a'dan e'ye kadar olan harfleri arar. sayıysa [2-14]

[^abc] abc dışındaki tüm karakterleri arar

[^0-9] rakam olmayanları arar

" . " bir basamaklı olanları (ya da tek harfleri) alır. nokta sayısı kadar basamak arar.

".." iki basamak olacak şekilde ayırır:

```
import re
txt = "The rain in Spain"
a = re.findall("..", txt)
print(a) >>> ['Th', 'e ', 'ra', 'in', ' i', 'n ', 'Sp', 'ai']
```

2. search() function searches the string for a match, and returns a Match object if there is a match. If there is more than one match, only the first occurrence of the match will be returned:

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

re.search() method searches the string to see if it starts with "The" and ends with "Spain":



```
import re
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

3. split() function returns a list where the string has been split at each match:

```
import re
txt = "The rain in Spain"
x = re.split("\s", txt)
print(x) >>> ['The', 'rain', 'in', 'Spain']
```

you can control the number of occurrences by specifying the maxsplit parameter:

```
import re
txt = "The rain in Spain"
x = re.split(" ", txt)
print(x) >>> ['The', 'rain', 'in', 'Spain']
```

split'e 3. bir parametre (1) ekledik.

4. sub() function replaces the matches with the text of your choice which replaces every boşluk with " \* " :

```
import re
txt = "The rain in Spain"
x = re.sub(" ", "*", txt)
print(x) >>> The*rain*in*Spain
```

- tkinter: Tkinter, Python kurulumu ile birlikte gelen ve pencere-li-menü-lü modern programlar yazmamızı sağlayan grafik arayüz geliştirme takımlarından biridir. Elbette Python'da grafik arayüzlü programlar yazmamızı sağlayacak tek modül Tkinter değildir. Bunun dışında PyQt, PyGI ve Kivy gibi alternatifler de bulunur. Ancak Tkinter'in öteki alternatiflere karşı en büyük üstünlüğü hem öbürlerine kıyasla çok daha kolay olması hem de Python'la birlikte gelmesidir.

mesela şu tkinter komutu boş bir pencere açar:

```
import tkinter as tk

#instantiation of tkinter module under Tk() class
pencere = tk.Tk()

pencere.mainloop()
```

aslında 2. kod satırında pencere oluşturulmuş oluyor ama mainloop()'u çağırmadıkça görülmüyor ekranda

eğer bu pencerenin boyutlarını ayarlamak istiyorsan tkinter'ın geometry() modülünü kullanmalısın:

```
import tkinter as tk

#instantiation of tkinter module under Tk() class
pencere = tk.Tk()
pencere.geometry("500x500")

pencere.mainloop()
```

bu boş pencereye bir etiket bir de düğme ekleyelim:

```
import tkinter as tk

instantiation of tkinter module under Tk() class
pencere = tk.Tk()

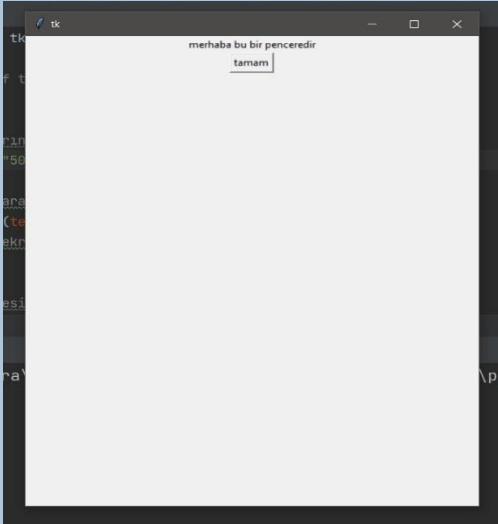
ekranın boyutlarını ayarladık
pencere.geometry("500x600")

etiket'e text parametresiyle bir yazı yazdırdık
```

```
etiket = tk.Label(text= "merhaba bu bir penceredir")
pack komutuyla ekrana yansıttık (bi nevi çalıştırdık kodu)
etiket.pack()

comman parametresiyle işlem atadık (tamam tıklayınca kapat)
düğme = tk.Button(text='tamam', command = pencere.destroy) # not destroy()
düğme.pack()

pencere.mainloop()
yani Label() metodu etiket (yazı) oluşturuyor Button() metodu düğme oluşturuyor pencerede
```



title() metodu pencerenin başlığını isimlendirir:

```
import tkinter as tk
from tkinter import *

genel bir tkinter çerçevesi yaratıyoruz
örnek = Tk()

başlıktaki yazıyı ayarlıyoruz
örnek.title("pencerenin başlığı")

pencere boyutunu ayarlıyoruz
root.geometry("300x300")

pencereyi çalıştıran kod
root.mainloop()
```

tkinter ekranından yazı yazdırarak çıkan kod:

```
import tkinter as tk

pencere = tk.Tk()
pencere.geometry("500x500")

pencereyi kapatmaya yarayan bir kod
çalıştığında etiketin ve düğmenin textini değiştiriyor
düğmenin durumunu disabled yaparak basılmaz hale getiriyor
3000 milisaniye sonra da otomatik kapatıyor ekranı
def Çıkış():
 etiket['text'] = 'Elvede pencere...'
 düğme['text'] = 'lütfen bekleyin...'
 düğme['state'] = 'disabled'
 pencere.after(3000, pencere.destroy)
```

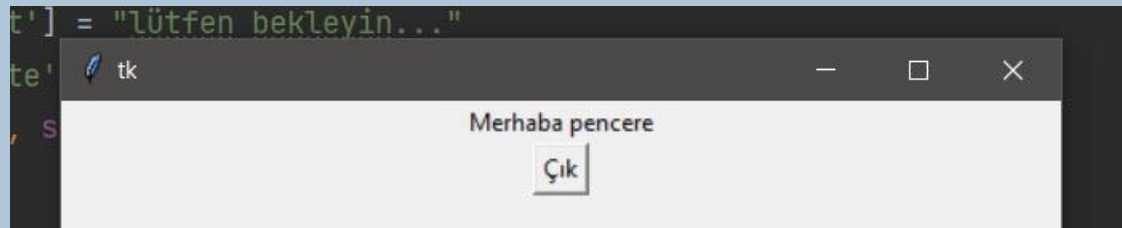
```
etiket = tk.Label(text = 'Merhaba Pencere')
etiket.pack()
```

```
düğme = tk.Button(text = "Çık",command = Çıkış)
düğme.pack()
```

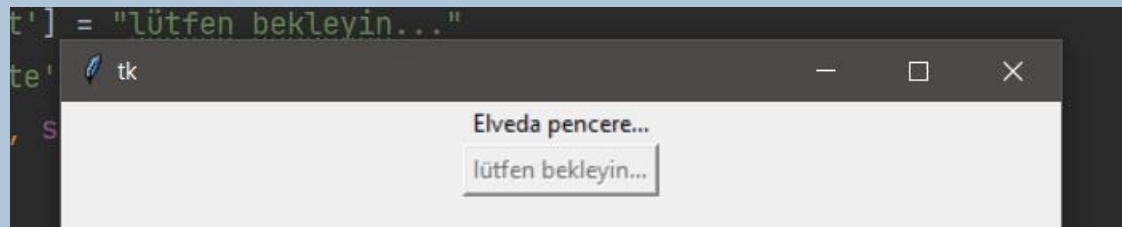
```
çarpıya basarak da kapatılabilsin diye Tk'nin protokol metoduyla da basıldığında
çıkış fonksiyonunu devreye sokacak bir kod yazdık
pencere.protocol("WM_DELETE_WINDOW",Çıkış)
```

```
pencere.mainloop()
```

bu kodlarda, satır sıraları çok önemlidir. Mesela burada düğmeyi oluşturan kodlarla `pencere.protocol()` kodlarının çalışması için bunların mutlaka `çıkış()` fonksiyonu tanımlandıktan sonra yazılması gerekir.



çık'a ya da çarpıya basarsak 3 saniye sonra çıkıyor



yukarıdaki kodu procedural programming paradigmasına göre yazdık. peki OOP paradigmasında nasıl yazarız:

```
import tkinter as tk
```

```
tk modülündeki Tk sınıfını miras aldık kendi __init__() metodumuzu tanımlarken, Tk()
sınıfının kendi __init__()
metodundaki işlemleri de gölgelemememiz lazım. Dolayısıyla orijinal __init__() metodunu
kendi __init__() metodumuza
aktarmak için super() kullandık
```

```
class Pencere(tk.Tk):
```

```
 def __init__(self):
 super().__init__()
 self.protocol("WM_DELETE_WINDOW", self.Çıkış)
```

```
 self.etiket = tk.Label(text = "Merhaba pencere")
 self.etiket.pack()
```

```
 self.düğme = tk.Button(text= "Çık", command = self.Çıkış)
 self.düğme.pack()
```

```
 def Çıkış(self):
 self.etiket['text'] = "Elveda pencere..."
 self.düğme['text'] = "lutfen bekleyin..."
 self.düğme['state'] = 'disabled'
 self.after(3000, self.destroy)
```

```
pencere = Pencere()
pencere.geometry("500x500")
pencere.mainloop()
```

## iterators

- An iterator is an object that contains a countable number of values. in Python, an iterator is an object which implements the iterator protocol. Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from. All these objects have a iter() method which is used to get an iterator:

```
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)
print(next(myit))
print(next(myit))
print(next(myit)) >>> alt alta apple banana cherry
```

iterator protocol consists of:

5. \_\_iter\_\_()
6. next()

even strings are iterable objects, and can return an iterator:

```
str = "formidable"
a = iter(str)
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a))
print(next(a)) >>> alt alta f o r m i d a b l e
```

- To create an object/class as an iterator you have to implement the methods \_\_iter\_\_() and \_\_next\_\_() to your object. as we know, all classes have a function called \_\_init\_\_(), which allows you to do some initializing when the object is being created. The \_\_iter\_\_() method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself. The \_\_next\_\_() method also allows you to do operations, and must return the next item in the sequence.

```
class Sayılar:
 def __iter__(self):
 self.a = 1
 return self
 def __next__(self):
 x = self.a
 self.a += 1
 return x
deneme = Sayılar()
myiter = iter(deneme)
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter)) >>> 4 kere yazdırdığımız için alt alta 1 2 3 4
```

bu kodu loop'a alalım ve sonsuza kadar gitmesin diye de bir condition belirleyelim. bunun için StopIteration metodunu kullanıyoruz:

```
class Sayılar:
 def __iter__(self):
 self.a = 1
 return self
 def __next__(self):
 if self.a ≤ 20:
 x = self.a
 self.a += 1
 return x
 else:
 raise StopIteration

deneme = Sayılar()
myiter = iter(deneme)
for i in myiter:
 print(i) >>> alt alta 1 2 3 4 ... 20
```

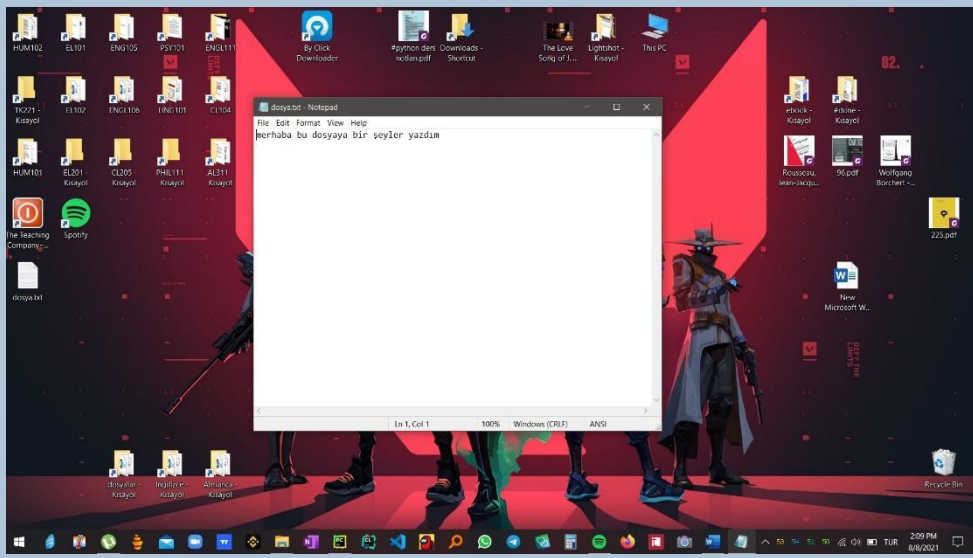
#file\_handling

- dosya açmak ya da oluşturmak için open() metodu kullanılır. şeması:  
`open(dosya_adi, dosyayı_açma_amacımız)`
- File handling is an important part of any web application. Python has several functions for creating, reading, updating, and deleting files. The key function for working with files in Python is the open() function. The open() function takes two parameters; dosyaAdı ve dosyayıAçmaAmacımız. There are 5 different methods for opening a file:
  - I. "w" - Write - Opens a file for writing, creates the file if it does not exist. yani bu parametreyle dosyayı açtığımız zaman yazma modunda açmış oluyoruz, yani dosyaya bir bilgi ekleyeceğiz. dosyayı mevcut konumda oluşturur.
  - II. "a" - Append - Opens a file for appending, creates the file if it does not exist. dosya konumda yoksa oluşturur.
  - III. "x" - Create - Creates the specified file, returns an error if the file exists. dosya zaten yoksa hata verir.
  - IV. "r" - Read - Default value. Opens a file for reading, error if the file does not exist. dosya konumda yoksa hata verir.
  - V. "r+" - Read & Write - you can both edit and read the content of a file.
  - VI. "w+" - Write & Read - same as r+ but if the file does not exist, a new one is made. Otherwise, the file is overwritten.
  - VII. "a+" - Append & Read - appending and reading mode. Similar to w+ as it will create a new file if the file does not exist. Otherwise, the file pointer is at the end of the file if it exists.

|                                                  | r | r+ | w | w+ | a | a+ |
|--------------------------------------------------|---|----|---|----|---|----|
| Read                                             | ✓ | ✓  | ✗ | ✓  | ✗ | ✓  |
| GoalKicker.com - Python® Notes for Professionals |   |    |   |    |   |    |
| Write                                            | ✗ | ✓  | ✓ | ✓  | ✓ | ✓  |
| Creates file                                     | ✗ | ✗  | ✓ | ✓  | ✓ | ✓  |
| Erases file                                      | ✗ | ✗  | ✓ | ✓  | ✗ | ✗  |

**I -** "w" modu python'la bir dosya yaratmak için kullanılır:

```
open("yaratilan_dosya.txt", "w")
dosyayla işin bitince kapaman gerekir:
file = open("merhaba.py", "w")
file.close()
istediğin bir konumda dosya oluşturmak için:
file = open("C:/users/bugra/desktop/dosya.txt", "w")
"w" moduyla yazdığın her satır dosyayı overwrite eder, yani bir önceki işlemi silip yazar.
write() metoduyla da dosya içine istediklerini yazabilirsin:
file = open("C:/users/bugra/desktop/dosya.txt", "w")
file.write("merhaba bu dosyaya bir şeyler yazdım")
sonuç:
```



encoding= metodunu dosya türkçe karakterleri göstermiyorsa kullanabilirsin. utf-8 kullanman gerek türkçe için:

```
file = open("C:/users/bugra/desktop/dosya.txt","w", encoding="utf-8")
file.write("buğra kara")
```

**II -** "a" modu dosyaya bir şeyler eklemek için kullanılır:

```
file = open("C:/users/bugra/desktop/dosya.txt","a")
file.write("bugra")
file.write(" kara") >>> bugra kara
```

"w" metodunda olduğu gibi dosya içeriği silinmez, varolana ekleme yapar.

örnek:

```
name = input("please enter your name:")
with open("guest.txt", "a", encoding="utf-8") as g:
 g.write(name)
```

kullanıcının input olarak girdiği isim guest isimli bir .txt dosyasına yazdırılır

**III -** "x" modu boş bir dosya yaratmak için kullanılır:

```
file = open("newfile2.txt", "x", encoding= "utf-8")
```

**IV -** "r" modu read modudur ve default olarak geçerlidir:

```
file = open("newfile.txt")
```

mesela deneme\_dosya.txt isminde bir dosyamız olsun (işlem yaptığımız Python dosyasıyla aynı klasörde olmak zorunda). To open this deneme\_dosya file, use the built-in open() function. The open() function returns a file object, which has a read() method for reading the content of the file:

```
file = open("deneme_dosya.txt", "r")
print(file.read()) >>> metin belgesinin içinde ne varsa onu yazdırır
```

to open a file for reading it is enough to specify the name of the file:

```
f = open("file_name.txt")
```

ya da aynı işi yapan bir başka kod:

```
f = open("file_name.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them. Make sure the file exists, or else you will get an error.

bir dosyanın içindeki bilgileri okumanın 5 yolu:

1. for metoduyla:

```
file = open("newfile.txt","r", encoding="utf-8")
for i in file:
 print(i, end="") >>>
merhaba
benim adim Bugra
19 yasındayim
```

2. read() metoduyla:

```
file = open("newfile.txt","r")
print(file.read()) >>> same
```

read()'in içine kaç yazarsan ilk o kadar karakteri yazdırır. ilk print çalıştığı zaman cursor yani | 10. karaktere gelince durur ve `close.file()` demediğimiz için orada kalır. daha sonra tekrar printle çağırdığımızda cursor kaldığı yerden devam eder.

3. `readline()` metoduyla: satır satır yazdırır .txt'i:

```
file = open("newfile.txt","r")
print(file.readline(), end= "")
print(file.readline(), end= "")
print(file.readline(), end= "") >>>
merhaba
benim adim Bugra
19 yasindayim
```

4. for looplarla:

```
a = "denemeDosya.txt"
with open("denemeDosya.txt") as dd:
 for line in dd:
 print(line.rstrip()) >>>
evden işe
işten eve
sökülmez
alamanya gurbetinin
dilleri
```

5. `readlines()` method:

```
a = "denemeDosya.txt"
with open("denemeDosya.txt") as dd:
 print(dd.readlines()) >>> ['evden işe \n', 'işten eve \n', 'sökülmez\n', 'alamanya
gurbetinin\n', 'dilleri']
```

**V -** `r+` method: "`r+`" modu bir dosyada güncelleme yapmak için kullanılır. yani hem okuma hem yazma:

```
with open("newfile.txt","r+") as file:
 print(file.read()) >>>
merhaba
benim adim Bugra
19 yasindayim
```

buna mesela "selamlar" stringini baştan eklemek için:

```
with open("newfile.txt","r+") as file:
 file.write("selamlar")
with open("newfile.txt","r+") as file:
 print(file.read()) >>>
selamlar
adim Bugra
19 yasindayim
```

- with method:

```
file = open("deneme.txt","r")
print(file.read())
```

ile bu aynı şeydir:

```
with open("deneme.txt","r") as file:
 print(file.read())
```

notice that you have to indent print line when used with method

- `write()` metodu girdiğin string'i bir dosyaya yazdırır:

```
text = '''Though wise men at their end know dark is right,
Because their words had forked no lightning they
```



```
Do not go gentle into that good night.\n'''
```

```
with open('thomas.txt', 'w') as f:
 f.write(text)
```

ya da ne yazacağına kod satırında kendin de karar verebilirsin:

```
with open('deneme.txt', 'w') as metinBelgesi:
 metinBelgesi.write("bu\nbir\ndeneme\nyazısıdır.")
```

- If the size of the file is tiny, it is safe to read the whole file contents into memory. If the file is very large it is often better to read line-by-line or by chunks, and process the input in the same loop. To do that:

```
with open('deneme.txt', 'r') as metinBelgesi:
 lines = []
 for line in metinBelgesi:
 lines.append(line.strip())
```

- Write a while loop that prompts users for their name. When they enter their name, print a greeting to the screen and add a line recording their visit in a file called guest\_book.txt. Make sure each entry appears on a new line in the file:

```
dosya = "guest.txt"
print("lütfen bitirmek için q yazınız:")
while True:
 name = input("lütfen isminizi giriniz:")
 if name == "q":
 break
 else:
 with open("guest.txt", "a", encoding="utf-8") as dd:
 dd.write(f"{name}\n")
 print(f"hi {name}, isminiz dosyaya eklendi")
```

- tell() metodu ise o anki cursor'un indeks olarak konumu verir:

```
fileobj = open('deneme.txt', 'r')
pos = fileobj.tell()
print('We are at %u.' % pos) >>> We are at 0.
```

herhangi bir okuma işlemi yaptıktan sonra cursor okunan karakter sayısı kadar index ilerler. mesela dosyanın içindeki karakter sayısı 22 ise tell() metodu 22 verecektir. daha sonra seek() içine yazacağın indeks ile cursor'ı oraya getirebilirsin.

- seek() metodu ise cursor'u istediğin yere konumlandırır:

```
file.seek(3)
```

cursor 3. indekse gitti, yani okuma işlemi girersen 3. indeksten itibaren okumaya başlayacaktır.

örnek :

```
with open('belge.txt', 'r') as f:
 contents = f.read()
 print(contents) >>>
```

```
arka
kapak
stefan
zweig
123456789
```

bu belgede tell() ve seek() metodunu kullanalım:

```
with open('belge.txt', 'r') as f:
 print(f.read(10))
 print(f"belge dosyasındaki cursor konumu {f.tell()}")
 f.seek(0,0)
 print(f"belge dosyasındaki cursor konumu {f.tell()}") >>>
```

```
arka
kapak
```

```
ste
belge dosyasındaki cursor konumu 11
belge dosyasındaki cursor konumu 0
```

1. `f.read(14)` metoduyla 14. karaktere kadar yazdırdık, dolayısıyla cursor 14'te durdu.
  2. `tell()` metoduyla konumunu sorunca 11 dedi yani 10u okuyup 11de durmuş.
  3. daha sonra seek metoduyla 0. byte'a geri gönderdik, bir nevi sıfırladık
- her bir karakter programlama dillerinde 1 byte'a denk gelir. bir şeyin memoryde ne kadar yer kapladığını öğrenmek için:

```
import sys
print(sys.getsizeof("a")) >>> 50
```

her bir karakter eklediğinde +1 kb eklenir.

- inputs can also be read from files. files can be opened using the keyword `open`.
- There are 3 basic I/O connections:
  1. standard input (`sys.stdin`): comes from a keyboard. The standard input and output in Python are objects located in the `sys` module.
  2. standard output (`sys.stdout`): Standard output is where we print our data with the `print` keyword. Unless redirected, it is the terminal console.
  1. standard error (`sys.stderr`): a stream where programs write their error messages. It is usually the text terminal.
- So far, we have been working with simple textual data. What if we are working with objects rather than simple text? For such situations, we can use the `pickle` module. This module serializes Python objects. The Python objects are converted into byte streams and written to text files. This process is called pickling. The inverse operation, reading from a file and reconstructing objects is called deserializing or unpickling.
- for instance to create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

- Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

- Python can only write strings to a text file. If you want to store numerical data in a text file, you'll have to convert the data to string format first using the `str()` function.
- To delete a file, you must import the `OS` module, and run its `os.remove()` function. for instance to delete a file named `demofile.txt` you must code:

```
import os
os.remove("demofile.txt")
```

- To avoid getting an error, you might want to check if the file exists before you try to delete it:

```
import os
if os.path.exists("demofile.txt"):
 os.remove("demofile.txt")
else:
 print("The file does not exist")
```

- To delete an entire folder, use the `os.rmdir()` method:

```
import os
os.rmdir("myfolder")
```

You can only remove empty folders.

- When users close a program, you'll almost always want to save the information they entered. A simple way to do this involves storing your data using the `json` module. The `json` module allows you to dump simple Python data structures into a file and load the data from that file the next time the program runs. You can also use `json` to share data between different Python programs. Even better, the JSON data format is not specific to Python, so you can share data you store in the JSON format with people who work in many other programming languages. It's a useful and portable format, and it's easy to learn. `json` module helps you to save users data so it won't be lost when the program stops running.

json'la uğraşırken import json demen gerekir. JSON is text, written with JavaScript object notation for storing and exchanging data.

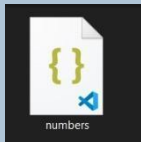
- The JSON (JavaScript Object Notation) format was originally developed for JavaScript. However, it has since become a common format used by many languages, including Python.
- json modülünün bilgisayarda nerde olduğunu görmek için:

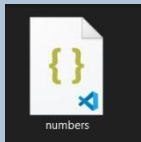
```
import json
print(json.__file__)
```

- open() function needs one argument: the name of the file you want to open. To do any work with a file, even just printing its contents, you first need to open the file to access it.
- json.dump() and json.load() usage. Let's write a short program that stores a set of numbers and another program that reads these numbers back into memory. The first program will use json.dump() to store the set of numbers, and the second program will use json.load().
- json.dump() function takes two arguments: a piece of data to store and a file object it can use to store the data. Here's how you can use json.dump() to store a list of numbers:

```
import json
numbers = [1, 42, 200, 10, -25, 13]
dosya = 'numbers.json'
with open(dosya, "w") as a:
 json.dump(numbers, a)
```

1. öncelikle import metoduyla json modülünü aktardık
2. sonra sayılardan oluşan numbers isminde bir liste tanımladık
3. .json uzantılı bir dosya yaratmak için bir değişken tanımlayıp adına numbers.json dedik
4. "w" moduyla with open() metodunu kullandık
5. daha sonra json.dump metoduyla a alias'ı verdiğimiz dosyaa json.dump metoduyla numbers'ı attık



6. klasörde  şeklinde bir dosyamız oluştu. tıklayınca listeyi veriyor. json.load() metoduyla da bunun tam tersini yapıyoruz, var olan bir .json dosyasından IDE'ye veri alıyoruz. json.load() metodu tek bir parametre alır:

```
import json
yeniDosya = 'numbers.json'
with open(yeniDosya) as yd:
 yeniListe = json.load(yd)
print(yeniListe) >>> [1, 42, 200, 10, -25, 13]
```

7. import'la json modülünü aktardık
8. içine aktarmak istediğimiz (boş) bir dosya tanımladık .json uzantılı
9. with open()'la dosyayı açtık
- 10.yeni bir değişken ismi tanımlayıp (yeniListe) json.load() metoduyla içine alias'ını yazdığımız verileri bu listeye aktardık
- 11.değişkeni yazdırdık

This is a simple way to share data between two programs.

- json.loads() metodu bir Json string'ini çözümlemene yarar. yani Json string'i Python Dict'e çevirir.

```
import json
x = '{ "name":"John", "age":30, "city":"New York"}'
y = json.loads(x)
print(y["age"]) >>> 30
```

json.dumps() metoduyse tam tersi. Python'dan Json'a çevirir:

```
a Python object (dict):
```

```
x = {
 "name": "John",
 "age": 30,
 "city": "New York"
}
```

```
convert into JSON:
```

```
y = json.dumps(x)
```

```
the result is a JSON string:
```

```
print(y)
```

- tüm veri tipleri Json'a çevrilebilir. yani python bir ifadeyi Javascript bir ifadeye çeviriyoruz

- indent= metoduyla ifadeyi okuması kolay hâle getirebilirsin:

```
json.dumps(x, indent=4)
```

- Use the sort\_keys parameter to specify if the result should be sorted or not:

```
json.dumps(x, indent=4, sort_keys=True)
```

- bir kullanıcıdan veri alıp bu veriyi bir daha girdiğinde hatırlayan kod:

```
import json
username = input("who are you? :")
dosyaİsmi = 'username.json'
with open(dosyaİsmi,"w") as d:
 json.dump(username, d)
 print(f"I'll remember you when you come back, {username.title()}!")
```

- o input aldık

- o dosyaİsmi'ne .json uzantılı dosyanın ismini girdik

- o json.dump ile input'la alınan username'yi d alias'lı dosyaİsmi .json'ına attık

- o print ile yazdırdık

daha sonra bu dosyayı kullanarak verilere tekrar erişelim:

```
import json
dosyaİsmi = 'username.json'
with open(dosyaİsmi) as d:
 username = json.load(d)
 print(f"hoşgeldin {username.title()}!")
```

- o dosyayı with open() ile açtık ve d alias'ı verdik

- o username değişkenine json.load metodunu kullanarak d dosyasının içeriğini attık (input'a girdiğini)

- o print ile bu içeriği yazdırdık

birleştirelim:

```
import json
username = input("who are you? :")
dosyaİsmi = 'username.json'
with open(dosyaİsmi,"w") as d:
 json.dump(username, d)
 print(f"I'll remember you when you come back, {username.title()}!")
username = input("who are you? :")
if username == "buğra":
 with open(dosyaİsmi) as d:
 username = json.load(d)
 print(f"hoşgeldin {username.title()}!") >>>
```

```
who are you? :buğra
```

```
I'll remember you when you come back, Buğra!
```

```
who are you? :buğra
```

```
hoşgeldin Buğra!
```

- Often, you'll come to a point where your code will work, but you'll recognize that you could improve the code by breaking it up into a series of functions that have specific jobs. This process is called refactoring.

- örnek:

```
import json
number = input("what's your favorite number? ")
with open("favorite_numbers.json","w") as fn:
 json.dump(number, fn)
 print("thanks, I'll remember that!")
```

and then

```
import json
with open("favorite_numbers.json") as fn:
 number = json.load(fn)
print(f"your favorite number is {number}!")
```

- python'da bir packet indirmek için terminalde: pip install paketin\_ismi yazmalısın. yüklü olan paketleri görmek için de pip list.
- \_\_file\_\_ metodu bir modülün dosya konumu görmek için kullanılır:

```
import json
print(json.__file__)
```

-

## #request

- request modülüyle internet sitelerinin html bilgilerine yani f11'e basınca çıkan kaynak kodlarına ulaşabiliriz. pip install ile indirmelisin bu modül zira built-in değildir. mesela jsonplaceholder.typicode.com'a ulaşalım:

```
import requests
a = requests.get("https://jsonplaceholder.typicode.com/todos")
print(a) >>> <Response [200]>
```

200 kodunu göndermesi everything's ok demektir. linki yanlış yazarsan 404 kodu gönderir.

- sayfadaki json bilgileri txt olarak almak istiyorsan da .text metodunu kullanmalısın:

```
import requests
a = requests.get("https://jsonplaceholder.typicode.com/todos")
a = a.text
print(a)
```

```
▼ 0:
 userId: 1
 id: 1
 title: "delectus aut autem"
 completed: false
▼ 1:
 userId: 1
 id: 2
 title: "quis ut nam facilis et officia qui"
 completed: false
▼ 2:
 userId: 1
 id: 3
 title: "fugiat veniam minus"
 completed: false
▼ 3:
 userId: 1
 id: 4
 title: "et porro tempora"
 completed: true
```

sayfada şöyle indeksli bilgiler var (dictionary). bu indekslerden belirli bir bilgiyi (mesela title) almak için dictionary mantığı kullanmamız gerekir ama bunun için önce json'a çevirmemiz lazım. sadece title yazan kısımların karşılıklarını almak için json.loads() metodunu kullanıyoruz:

```
import requests
import json
a = requests.get("https://jsonplaceholder.typicode.com/todos")
a = json.loads(a.text)
for i in a:
 print(i["title"])
```

- api: application programming interface. bir uygulamanın, servisin ve/veya platformun sahip olduğu yeteneklere izin verilen sınırlandırmalar dahilinde dışarıdan erişilebilmesini sağlayan bir arayüzdür. API(Application Programming Interface) bizim dilimizde "Uygulama Programlama Arayüzü", bir uygulamanın işlevlerine dışarıdan veya uzaktan erişilip bu işlevlerin kullanılmasını sağlayan arayüzdür. API, bir sunucunun üzerindeki uygulamaya farklı platformlardan ulaşılmasını ve response dönmesine olanak sağlar. Web API'lerinin tamamı REST(REpresentational State Transfer) mimarisi üzerinde dizayn edilir. Bundan dolayı platform bağımsız çalışır. Bu mimari GET, POST, PUT, DELETE metodlarının hepsini desteklemektedir. Web API çıktıları talebe göre JSON, XML gibi çeşitli çıktıları olabilir. API kullanımının asıl amacı bir uygulamanın bütün veya bazı metodlarını diğer uygulamalara kullanıma açarak uzaktan gelecek veri ve bilgi taleplerini kolayca ve hızlıca karşılamaktır. Böylelikle tek bir uygulamada gerçekleşen işlemlerden izin verilen uzak kullanıcılar belirli parametreler sayesinde faydalanabileceklerdir. API genel olarak gerçek zamanlı veriyi tek tek işlemeye yarar. Sunucunun API üzerinden gönderdiği parametre içeren veya içermeyen girdiyi sunucu işler ve geriye bir sonuç kümesi veya sadece başarı bildirimi döner. Verinin sadece belli bir kısmında yapılacak güncellemeler bir parametre gerektirir. API ise bu işlemlerin hem hızlı hemde pratik olmasını sağlar. Entegrasyon tarafına gelecek olursak, karşıdaki sunucunun API üzerinden izin verdiği fonksiyonları

kullanabilmek için öncelikle istemciyi tanıtacak bir key gereklidir. Daha sonra bu key ile kullanılabilecek erişimi onaylayacak olan bir şifre almak gerekir. API hizmeti veren sunucu tarafındaki kurum, ilk olarak istemcinin başvurusunda key ve şifreyi ister. İstenen fonksiyonlar kullanılır ve istenen bilgiler karşı uygulamadan API'lerin döndüğü response'lar vasıtasıyla alınır. İsteğe göre API'lara erişim ile ilgili kısıt konulabilir, bunun nedeni API'leri aşırı meşgul olmasını istememeden kaynaklıdır. Bu kullanım sürecinde uygulamanın kendisine ihtiyaç duymadan sunulan özellikleri, fonksiyonları, içerikleri edinebilir ve/veya gönderim yapabilirsiniz. Elbette -çoğunlukla- erişimler belirli sınırlandırmalara sahiptir ve loglanırlar. Ayrıca, gerçekleştirmek istediğiniz işlemler için erişim sağlamak istediğiniz uygulama tarafından size özel sunulan erişim bilgilerini kullanmanız gerekir. web services olarak ifade edilen iki tip yaygın olarak kullanılmakta:

- o SOAP: simple object access protocol. web servis çağrılarında RPC (Remote Procedure Call) kullanır. istemci/sunucu mantığına dayanır. Metodların call edilmesi üzerine alınan response da dahil sürecin tamamı XML olarak ve genellikle HTTP (Hyper Text Transfer Protocol) protokolü (bazende TCP/IP) kullanılarak iletilir. XML yapısı sebebiyle REST'e kıyasla daha katı bir işleyişe sahip olduğu için pek tercih edilmemektedir.
- o REST: representational state transfer

•

## #scopes

- there are two scopes in python:

1. Global
2. Local

### 1 - Global

A variable or function that is created in the main body of a Python code is called global (variable/function) and is part of the global scope:

```
greet = "merhaba zalim dünya"
def deneme():
 print(greet)
deneme()
merhaba zalim dünya
```

global metodu: Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the global keyword. If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
 global x
x = "fantastic"
myfunc()
print("Python is " + x)
```

global scope örnek:

```
def myfunc():
 global x
 x = 300
myfunc()
print(x) >>> 300
```

global keyword'ünün diğer bir kullanım alanı fonksiyon içindeki global variable'da bir değişiklik yapmak içindir:

```
x = 300
def myfunc():
 global x
 x = 200
myfunc()
print(x) >>> 200
```

global scope örnek:

```
x = 300
def myfunc():
 print(x)
myfunc()
print(x) >>> 300 \n 300
```

### 2 - Local Scope

A v/f created inside a function is called a local v/f and is a part of the local scope:

```
def deneme():
 greet = "merhaba zalim dünya"
 print(greet)
deneme()
merhaba zalim dünya
```

greet fonksiyon blogunun içinde yazıldığı için sadece orada geçerlidir. satır başı print(greet) diyemezsin yani.

A variable defined in a function body has a local scope. It is valid only within the body of the function:



```
name = "Jack"
def isimler():
 name2 = "Tony"
print(name)
print(name2) >>>
Jack
name2 is not defined
```

local scope örnek:

```
def myfunc():
 x = 300
 print(x)
myfunc() >>> 300
```

that x variable is local.

however it is acceable for the inner function of the function that x variable belongs to:

```
def myfunc():
 x = 300
 def innerfunc():
 print(x)
 innerfunc()
myfunc()
```

- eğer aynı isimde biri global biri local iki farklı değişken tanımlarsan python onları farklı variables addedecektir:

```
x = 300
def myfunc():
 x = 200
 print(x)
myfunc()
print(x) >>> 200 \n 300
```

Scopes matter for several reasons:

- Code in the global scope, outside of all functions, cannot use any local variables.
- However, code in a local scope can access global variables.
- Code in a function's local scope cannot use variables in any other local scope.
- You can use the same name for different variables if they are in different scopes. That is, there can be a local variable named spam and a global variable also named spam.

- global/local scope ayrımına dikkat etmezsen hata alırsın:

```
def fun():
 num = 1123
fun()
print(num)
```

bu kod hata verecektir çünkü local scope'ta tanımladığın bir num değişkenini global'de kullanmaya çalışıyorsun. printi fun blogunun altına yazmak zorundasın.

- ayrıca bir başka local scope'taki değişkenleri de yeni value atamak için kullanamazsın:

```
def fun():
 num = 1123
 funb()
 print(num)
def funb():
 num = 5532
fun() >>> 1123
```

- If you need to modify a global variable from within a function, use the global statement. If you have a line such as global eggs at the top of a function, it tells Python, "In this

function, num refers to the global variable, so don't create a local variable with this name." :

```
def fun():
 global num
 num = 1123

num = "global değişken"
fun()
print(num) >>> 1123
```

- There are four rules to tell whether a variable is in a local scope or global scope:
  - o If a variable is being used in the global scope (that is, outside of all functions), then it is always a global variable.
  - o If there is a global statement for that variable in a function, it is a global variable.
  - o Otherwise, if the variable is used in an assignment statement in the function, it is a local variable.
  - o But if the variable is not used in an assignment statement, it is a global variable.

- örnek:

```
def fun():
 global num
 num = 1123 # this is global

def funb():
 num = 4423 # this is local

def func():
 print(num) # this is global

num = 9941 # this is global

fun()
print(num) >>> 1123
```

- tips:
  - o global scope'taki kod local scop'ta tanımlanmış değişkenleri kullanamaz
  - o ama local scope'ta tanımlanmış bir değişken global scope'ta tanımlanmış değişkenleri kullanabilir
  - o local scope'taki bir değişken başka local scope'lardaki değişkenlere ulaşamaz
  - o local ve global scope içinde aynı isimli farklı değişkenler tanımlayabilirsiniz
-

## #ErrorHandling

- 5 temel hata vardır:
  - o `nameError` - `print(a)` diyip a'yı beforehand tanımlamadıysan
  - o `syntaxError` - `print("denem"e)` dersen
  - o `zeroDivisionError` - `print(10/0)`
  - o `valueError` - `int(1a3)`
  - o mantık hatası - eğer kod hata vermiyor ama eksik/yanlış çalışıyorsa (en zoru)
- The code that could potentially have an error is put in a try clause. The try-except-else block works like this: Python attempts to run the code in the try statement. The only code that should go in a try statement is code that might cause an exception to be raised. Sometimes you'll have additional code that should run only if the try block was successful; this code goes in the else block. The except block tells Python what to do in case a certain exception arises when it tries to run the code in the try statement. By anticipating likely sources of errors, you can write robust programs that continue to run even when they encounter invalid data and missing resources. Your code will be resistant to innocent user mistakes and malicious attacks.
- When an error occurs, or exception as we call it, Python will normally stop and generate an error message. These exceptions can be handled using the try statement. for instance, The try block will generate an exception, because x is not defined:

```
try:
 print(x)
except:
 print("An exception occurred") >>> An exception occurred
 o because x is not previously defined
 o Since the try block raises an error, the except block will be executed.
 o Without the try block, the program will crash and raise an error.
```

- mesela şöyle bir kod yazdık:

```
try:
 def zero(n):
 return 23 / n
 print(zero(0))
except:
 print("0'a bölemezsin") >>> 0'a bölemezsin
```

eğer `TypeError` verirse bir except bloğu daha yazabilirsin:

```
try:
 print(5/"k")
except ZeroDivisionError:
 print("0'a bölemezsiniz")
except TypeError:
 print("sayı girmek zorundasın")
```

bunları tek tek yazmak yerine parantez içinde virgülle de ayırabilirsin:

```
except (ZeroDivisionError, TypeError):
```

eğer hangi hata olduğunu öğrenmek istiyorsan da:

```
except (ZeroDivisionError, TypeError) as e:
```

mesela `5/0` `ZeroDivisionError` hatası verir. `5/"s"` girersen de `TypeError`.

hiç hata ismi yazmadan direkt `except:` de yazabilirsin.

- bir başka `zeroDivision` örneği:

```
def input_numbers():
 a = float(input("enter first number: "))
 b = float(input("enter second number: "))
 return a, b
a, b = input_numbers()
print(f"{a} / {b} is {a/b}")
```

eğer b'ye 0 girersen `zero division` hatası verir. bunu iki şekilde çözebilirsin:

```
def input_numbers():
 a = float(input("enter first number: "))
 b = float(input("enter second number: "))
 return a, b
a, b = input_numbers()
while True:
 if b != 0:
 print(f"{a} / {b} is {a / b}")
 break
 else:
 print("you cannot divide a float to Zero")
 a, b = input_numbers()
```

ya da try:except metoduyla:

```
def input_numbers():
 a = float(input("enter first number: "))
 b = float(input("enter second number: "))
 return a, b
a, b = input_numbers()
try:
 print(f"{a} / {b} is {a/b}")
except ZeroDivisionError:
 print("you can't divide by 0")
```

- print yerine raise keyword'ünü de kullanabilirsin. ValueError + raise kullanımı:

```
def read_age():
 age = int(input("please enter your age: "))
 if age < 0 or age > 130:
 raise ValueError("Invalid age!")
 return age
try:
 val = read_age()
 print(f"your age is {val}")
except ValueError as e:
 print(e)
```

- örnek:

```
try:
 a = (1, 2, 3, 4)
 print(a[5])
except IndexError as e:
 print(e)
 print("Class:", e.__class__)
```

except'ten sonra else: satırını eklersen except'lerin çalışmadığı (kodun hata vermediği) durumda o çalışır. ve sonunda da bir finally: bloğu gelir.

- The try block lets you test a block of code for errors.  
The except block lets you handle the error.  
The finally block lets you execute code, regardless of the result of the try- and except blocks.
- You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error. for instance print one message if the try block raises a NameError and another for other errors:

```
try:
 print(x)
except NameError:
 print("Variable x is not defined")
```

```
except:
 print("Something else went wrong")
```

- You can use the else keyword to define a block of code to be executed if no errors were raised:

```
try:
 print("Hello")
except:
 print("Something went wrong")
else:
 print("Nothing went wrong")
```

In this example try block does not give any error

- The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
 print(x)
except:
 print("Something went wrong")
finally:
 print("The 'try except' is finished")
```

- Try to open and write to a file that is not writable:

```
try:
 f = open("deneme_dosya.txt")
 f.write("Lorum Ipsum")
except:
 print("Something went wrong when writing to the file")
finally:
 f.close()
```

- örnek:

```
def check_password(psw):
 import re
 if len(psw) < 8:
 raise Exception("parola en az 7 karakterden oluşmalıdır.")
 elif not re.search("[a-z]", psw):
 raise Exception("parola küçük harf içermelidir.")
 elif not re.search("[A-Z]", psw):
 raise Exception("parola büyük harf içermelidir.")
 elif not re.search("[0-9]", psw):
 raise Exception("parola rakam içermelidir.")
 elif not re.search("[_&$@]", psw):
 raise Exception("parola alpha numeric karakter içerebilir.")
 elif re.search("\s", psw):
 raise Exception("parola boşluk içermemelidir.")
 else:
 print("geçerli parola")
password = "123456"
try:
 check_password(password)
except Exception as ex:
 print(ex) >>> parola en az 7 harften oluşmalıdır.
```

örnek2:

```
class Person:
 def __init__(self, name, year):
 if len(name) > 10:
 raise Exception("10 karakterden fazla bir isim girdiniz!")
```

```
else:
```

```
 self.name = name
```

```
p1 = Person("buğrahankaramollaoğlu", 2001) >>> Exception: 10 karakterden fazla bir isim girdiniz!
```

- bu listedeki sadece int'e çevrilebilen sayıları yazdıran program:

```
liste = ["1","2","5a","10b","abc","10","50"]
```

```
for i in liste:
```

```
 try:
```

```
 a = int(i)
```

```
 print(a)
```

```
 except ValueError:
```

```
 continue >>> 1 2 10 50
```

- sadece input verdiğinde değil inputu bir değişkene atadığında da input otomatik olarak çalışır:

```
a = input("hi?")
```

- q'ya basana kadar float isteyen float girmezsen hata veren program:

```
while True:
```

```
 sayi = input("sayi:")
```

```
 if sayi == "q":
```

```
 break
```

```
 try:
```

```
 result = float(sayi)
```

```
 print("girdiğiniz sayi:", result)
```

```
 except ValueError:
```

```
 print("geçersiz sayi")
```

```
 continue
```

while True: girdiğin bir kod blogunda mutlaka bir yerde break vermelisin yoksa sonsuza kadar gidecektir.

- girilen parola içinde türkçe karakter varsa türkçe karakter hatası veren program:

```
turkce_karakterler = ["ı","ğ","ş","ü","ö","ç",]
```

```
parola = input("parola giriniz:")
```

```
for i in parola:
```

```
 if i in turkce_karakterler:
```

```
 raise TypeError("parola türkçe içeremez.")
```

```
 else:
```

```
 pass
```

```
print("geçerli parola")
```

- faktoriyel hesaplamada hata oluşturma:

```
def faktoriyel(x):
```

```
 x = int(x)
```

```
 if x < 0:
```

```
 raise ValueError("x 0'dan küçük girilemez.")
```

```
 result = 1
```

```
 for i in range(1, x+1):
```

```
 result *= i
```

```
 return result
```

```
for a in [5, 6, 2, -3, "2a"]:
```

```
 try:
```

```
 y = faktoriyel(a)
```

```
 except ValueError as err:
```

```
 print("hata", err)
```

```
 continue
```

```
 print(y)
```

- yield keyword'üyle ram'de yer kaplamadan geçici bir fonksiyon tanımlarız. bu fonksiyon bellek üzerinde saklanmadığından bir daha çağırmak istesek çağıramıyoruz:

```
def cube():
 for i in range(5):
 yield i ** 3
for i in cube():
 print(i) >>> 0 1 8 27 64
```

- bu işleme generator denir. aynısını comprehensionla da yapabiliriz fakat generator olsun (bellekte yer kaplamasın) istiyorsak köşeli değil normal parantez kullanmamız gerekir:

```
liste = [i**3 for i in range(5)]
```

değil de

```
liste = (i**3 for i in range(5))
print(next(liste))
print(next(liste))
print(next(liste))
print(next(liste))
print(next(liste))
```

next() metodu gereği her bir print(next(liste)) satırı 1 iter'i (sırayla 0 1 8 27 64) yazdırır

- mesela fileHandling'le ilgili bir örnek:

```
a = 'yeni.txt'
with open('yeni.txt') as a:
 print(a.read()) >>> FileNotFoundError: [Error 2] No such file or directory: 'yeni.txt'
```

hata verdi çünkü böyle bir dosya yok, olsa bile .py dosyasıyla aynı klasörde değil. traceback vermesin diye try except metodunu kullanıyoruz:

```
a = 'yeni.txt'
try:
 with open('yeni.txt') as a:
 print(a.read())
except FileNotFoundError:
 msg = f"sorry, the file {a} does not exist."
 print(msg) >>> sorry, the file yeni.txt does not exist.
```

- .txt formatındaki bir kitaptaki kelime sayısını öğrenmek. 'when titans drive.txt' isminde bir metnimiz olsun. kelime sayısını öğrenmek için split() metodunu kullanıyoruz:

```
metin_ismi = "when titans drive.txt"
with open("when titans drive.txt", encoding="utf-8") as wtd:
 content = wtd.read()
words = content.split()
print(f"this book consists of {len(words)} words") >>> this book consists of 29628 words
```

- o dosyayı .py ile aynı klasöre attık
- o sonra dosya ismini bir değişkene (metin\_ismi) atadık
- o sonra with open() metoduyla dosyayı açıp utf-8'e ayarladık ve wtd diye bir alias tanımladık
- o sonra wtd metninde read() metodunu kullanıp content'e attık. yani bu metinde okunabilecek her şeyi content'le belirttik
- o sonra content'i split edip, yani tek tek boşluklardan ayırarak kelime haline getirip bir liste oluşturduk ve bu listeyi words değişkenine tanımladık
- o daha sonra words'ün eleman sayısını len() metoduyla yazdırdık.

- örnek:

```
print("çıkamak için q'ya basınız")
while True:
 try:
 num1 = int(input("bir sayı giriniz:"))
 if num1 == 'q':
```

```

 break
 num2 = int(input("bir sayı daha giriniz:"))
 if num2 == 'q':
 break
except ValueError:
 print("sayı girmelisiniz!")
else:
 sum = num1 + num2
 print(f"the sum of {num1} and {num2} is {sum}")

```

- iki farklı dosya içeriğini ekrana yazdırma:

```

dosyalar = ["motor.txt", "araba.txt"]
for i in dosyalar:
 print(f"reading file named {i}...")
 try:
 with open(i, encoding= "utf-8") as x:
 print(x.read())
 except FileNotFoundError:
 print("aradığınız metin dosyada yok :(") >>>
reading file named motor.txt...
harley
kawasaki
hyosung
reading file named araba.txt...
honda
qashqai
porsche

```

- eğer araba.txt metnini silersek except'teki hatayı verecektir. eğer hata vermesin istiyorsan pass ekler altına.
- eğer python spesifik bir hatayla karşılaştığında hata vermesin (silent fail) istiyorsan pass metodunu kullan:

```

try:
 """"xxx""""
except:
 pass
else:
 """"yyy""""

```



## #unit\_testing

- When you write a function or a class, you can also write tests for that code. Testing proves that your code works as it's supposed to in response to all the input types it's designed to receive. When you write tests, you can be confident that your code will work correctly as more people begin to use your programs. You'll also be able to test new code as you add it to make sure your changes don't break your program's existing behavior. Every programmer makes mistakes, so every programmer must test their code often, catching problems before users encounter them. In this chapter you'll learn to test your code using tools in Python's unittest module. You'll learn to build a test case and check that a set of inputs results in the output you want. You'll see what a passing test looks like and what a failing test looks like, and you'll learn how a failing test can help you improve your code.
- mesela basit bir isim + soyisim alıp bunları tek seferde yazdıran name\_function.py bir fonksiyon tanımlayalım:

```
def get_formatted_name(first,last):
 full_name = first + ' ' + last
 return full_name.title()
```

daha sonra bunu import edelim

```
from name_function import get_formatted_name
print("enter 'q' at any time to quit.")
while True:
 first = input("\nPlease give me a first name: ")
 if first == 'q':
 break
 last = input("Please give me a last name: ")
 if last == 'q':
 break
 formatted_name = get_formatted_name(first,last)
 print("\tNeatly formatted name: " + formatted_name + '.') >>>
```

```
Please give me a first name: cemre
Please give me a last name: kara
 Neatly formatted name: Cemre Kara.
Please give me a first name: q
Process finished with exit code 0
```

- unittest metodu is a module used when you want to test a code. A unit test verifies that one specific aspect of a function's behavior is correct A test case is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle. A good test case considers all the possible kinds of input a function could receive and includes tests to represent each of these situations.
- To write a test case for a function, import the unittest module and the function you want to test. Then create a class that inherits from unittest.TestCase, and write a series of methods to test different aspects of your function's behavior.
- The module unittest from the Python standard library provides tools for testing your code. A unit test verifies that one specific aspect of a function's behavior is correct. A test case is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle.

## #numpy

- NumPy is a Python library. NumPy is used for working with arrays. NumPy is short for "Numerical Python". It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray. Arrays are very frequently used in data science, where speed and resources are very important. NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.

- numpy ile python'da matematiksel verileri görsele döküyoruz. kullanmadan önce import et

```
import numpy as np
```

- np.array() metodu numpy array yaratmak için kullanılır:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5]) # parantez içinde de olabilir
print(arr) >>> 1 2 3 4 5
print(type(arr)) >>> <class 'numpy.ndarray'>
```

- numpy'ın versiyonunu öğrenmek için:

```
import numpy as np
print(np.__version__)
```

- mesela şöyle bir listemiz olsun:

```
import numpy as np
result = np.array([1,3,5,7,9]) >>> [1 3 5 7 9]
result = np.arange(1,11) >>> [1 2 3 4 5 6 7 8 9 10]
result = np.arange(1,21,2) >>> [1 3 5 7 9 11 13 15 17 19]
```

- Below is a list of all data types in NumPy and the characters used to represent them.
  - i - integer
  - b - boolean
  - u - unsigned integer
  - f - float
  - c - complex float
  - m - timedelta
  - M - datetime
  - O - object
  - S - string
  - U - unicode string
  - V - fixed chunk of memory for other type ( void )

- dtype metodu bir array nesnesinin tipini verir:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype) >>> int32
```

- bir başka örnek this time with strings:

```
import numpy as np
arr = np.array(['apple', 'banana', 'cherry'])
print(arr.dtype) >>> <U6
```

- dtype metodu ile veri tipini de belirleyebiliriz:

```
import numpy as np
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr) >>> [b'1' b'2' b'3' b'4']
print(arr.dtype) >>> |S1
```

the best way to change the data type of an existing array, is to make a copy of the array with the astype() method. The astype() function creates a copy of the array, and allows

you to specify the data type as a parameter. The data type can be specified using a string, like 'f' for float, 'i' for integer etc. or you can use the data type directly like float for float and int for integer.

- Change data type from float to integer by using 'i' as parameter value:

```
import numpy as np
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr) >>> [1 2 3]
```

- örnek2:

```
import numpy as np
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)
print(newarr) >>> [True False True]
```

- numpy'in şu metotları da vardır:

- o np.ones
- o np.zeros
- o np.linspace
- o np.random.randint
- o np.random.rand
- o np.array.reshape

- dimension of arrays (nested arrays): are arrays that have arrays as their elements. an array within an array so to speak. these are categorized as:

- o 0-D array: ilk seviye array
- o 1-D
- o 2-D
- o 3-D...

0-D array: normal array, içinde bir array olmayan:

```
import numpy as np
arr = np.array(43)
print(arr) >>> [43]
```

1-D array: içinde 0-D array olan arraya denir:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

2-D array: içinde 1-D array olan arraya denir. These are often used to represent matrix or 2nd order tensors. NumPy has a whole sub module dedicated towards matrix operations called numpy.mat:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr) >>>
[[1 2 3]
 [4 5 6]]
```

3-D array: içinde 2-D array olan arraya denir.

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr) >>>
[[[1 2 3]
 [4 5 6]]
 [[1 2 3]
 [4 5 6]]]
```

ndim metodu bir arrayın kaç boyutlu (dimension) olduğunu yazdırır:

```
import numpy as np
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
```

```
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(a.ndim) >>> 0
print(b.ndim) >>> 1
print(c.ndim) >>> 2
print(d.ndim) >>> 3
```

ayrıca bu metodla bir arrayin kaç boyutlu olacağına da karar verebilirsiniz:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
```

çok boyutlu arraylerde bir elemanı yazdırmak:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('5th element on 2nd dim: ', arr[1, 4]) >>> 5th element on 2nd dim: 10
```

access the third element of the second array of the first array:

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

- The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array. The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy. The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view. `copy()` metodunu kullanırız:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr) >>> [42 2 3 4 5]
print(x) >>> [1 2 3 4 5]
```

The copy SHOULD NOT be affected by the changes made to the original array. diğerini de `view()` metodu ile yapıyoruz:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr) >>> [42 2 3 4 5]
print(x) >>> [42 2 3 4 5]
```

The view SHOULD be affected by the changes made to the original array. ayrıca `view()` olarak tanımladığın arrayde yaptığın değişiklikler orijinal arrayi de değiştirir:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 30
print(arr) >>> [30 2 3 4 5]
print(x) >>> [30 2 3 4 5]
```

The original array SHOULD be affected by the changes made to the view.

- NumPy array has the attribute `base` that returns `None` if the array owns the data. `copies` owns the data, and `views` does not own the data. check if an array owns it's data or not:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
y = arr.view()
print(x.base) >>> None
print(y.base) >>> [1 2 3 4 5]
```

- The shape of an array is the number of elements in each dimension. NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements. mesela 2-D bir arrayin şeklini yazdıran program:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr) >>>
[[1 2 3 4]
 [5 6 7 8]]
print(arr.shape) >>> (2, 4)
```

bu şu demek; array has two dimensions, each dimension has four elements.

örnek:

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr) >>> [[[[[1 2 3 4]]]]]
print('shape of array :', arr.shape) >>> shape of array : (1, 1, 1, 1, 4)
```

- The shape of an array is the number of elements in each dimension. you can also reshape arrays. mesela 12 elemanlı 1-D bir arrayi 2-D bir arraya çevirelim:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print(arr) >>> [1 2 3 4 5 6 7 8 9 10 11 12]
newarr = arr.reshape(4, 3)
print(newarr) >>>
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [10 11 12]]
```

1-D tek satırda yazdırırken 2-D 3'er elemanlı 4 satırda yazdırdı. şimdi 1-D to 3-D yapalım:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

newarr = arr.reshape(2, 3, 2)

print(newarr) >>>
[[[1 2]
 [3 4]
 [5 6]]

 [[7 8]
 [9 10]
 [11 12]]]
```

one thing to notice, we can reshape an 8 elements 1D array into 4 elements in 2 rows 2D array but we cannot reshape it into a 3 elements 3 rows 2D array as that would require 3x3 = 9 elements.

- unknown dimension: You are allowed to have one "unknown" dimension. Meaning that you do not have to specify an exact number for one of the dimensions in the reshape method. Pass -1 as the value, and NumPy will calculate this number for you.

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
newarr = arr.reshape(2, 2, -1)
```

```
print(newarr) >>>
```

```
[[[1 2]
 [3 4]]
```

```
 [[5 6]
 [7 8]]]
```

- Flattening array means converting a multidimensional array into a 1D array. We can use `reshape(-1)` to do this:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
newarr = arr.reshape(-1)
```

```
print(newarr) >>> [1 2 3 4 5 6]
```

There are a lot of functions for changing the shapes of arrays in numpy `flatten`, `ravel` and also for rearranging the elements `rot90`, `flip`, `fliplr`, `flipud` etc. These fall under Intermediate to Advanced section of numpy.

- Iterating means going through elements one by one. As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python. If we iterate on a 1-D array it will go through each element one by one:

```
import numpy as np
```

```
arr = np.array([1, 2, 3])
```

```
for x in arr:
 print(x) >>> alt alta 1 2 3
```

Loop + 2-D:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
for x in arr:
 print(x)
```

- örnek:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
for x in arr:
 for y in x:
 print(y) >>> alt alta 1 - 6
```

örnek2:

```
import numpy as np
```

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

```
for x in arr:
 for y in x:
 for z in y:
 print(z) >>> alt alta 1-12
```

however, in basic for loops, iterating through each scalar of an array we need to use  $n$  for loops which can be difficult to write for arrays with very high dimensionality. we then may use `nditer()` function to simplify it:

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
 print(x)
```

- We can use `op_dtypes` argument and pass it the expected datatype to change the datatype of elements while iterating. NumPy does not change the data type of the element in-place (where the element is in array) so it needs some other space to perform this action, that extra space is called buffer, and in order to enable it in `nditer()` we pass `flags=['buffered']`:

```
import numpy as np

arr = np.array([1, 2, 3])

for x in np.nditer(arr, flags=['buffered'], op_dtypes=['S']):
 print(x) >>>
b'1'
b'2'
b'3'
```

- Iterate through every scalar element of the 2D array skipping 1 element:

```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

for x in np.nditer(arr[:, ::2]):
 print(x) >>> 1 3 5 7
```

- `ndenumerate()` method:

```
import numpy as np
arr = np.array([1, 2, 3])
for idx, x in np.ndenumerate(arr):
 print(idx, x)
>>>
(0,) 1
(1,) 2
(2,) 3
```

- bir numpy arrayindeki bir elemanın indeksini öğrenmek:

```
import numpy as np
numbers = np.array([5,10,15,20,25,30,35,40,45])
print(numbers[4]) >>> 25
```

normal indeks kuralları geçerlidir

```
import numpy as np
numbers = np.array([[5,10,15],[20,25,30],[35,40,45]])
print(numbers[0,2]) >>> 0. indeksteki 2. eleman → 15
```

bir diğer örnek

```
print(numbers[:, 2]) >>> [15 30 45] bütün elemanlardan 2. elemanı al
```

- `numpy random` kullanımına örnek:

```
import numpy as np
nums1 = np.random.randint(10, 100, 4)
nums2 = np.random.randint(10, 100, 4)
```

```
print(nums1) >>> [27 59 47 87]
print(nums2) >>> [87 76 46 81]
```

- bunları toplayalım:

```
import numpy as np

nums1 = np.random.randint(10, 100, 4)
nums2 = np.random.randint(10, 100, 4)

total = nums1 + nums2
print(total) >>> [107 139 122 115]
```

her bir elemana 10 eklemek için:

```
nums1 + 10
```

ayrıca np.sin/cos/sqrt/log() metotları ile matematiksel işlemler de uygulayabilirsiniz

- Joining means putting contents of two or more arrays in a single array. In SQL we join tables based on a key, whereas in NumPy we join arrays by axes. We pass a sequence of arrays that we want to join to the concatenate() function, along with the axis. If axis is not explicitly passed, it is taken as 0. joining to arrays:

```
import numpy as np
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
arrTotal = np.concatenate((arr1, arr2))
print(arrTotal) >>> [1 2 3 4 5 6]
```

- Join two 2-D arrays along rows (axis=1):

```
import numpy as np
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr) >>>
[[1 2 5 6]
 [3 4 7 8]]
```

- Stacking is same as concatenation, the only difference is that stacking is done along a new axis. We can concatenate two 1-D arrays along the second axis which would result in putting them one over the other, ie. stacking. We pass a sequence of arrays that we want to join to the stack() method along with the axis. If axis is not explicitly passed it is taken as 0:

```
import numpy as np
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=1)
print(arr) >>>
[[1 4]
 [2 5]
 [3 6]]
```

- NumPy provides a helper function: hstack() to stack along rows:

```
import numpy as np

arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])

arr = np.hstack((arr1, arr2))

print(arr) >>> [1 2 3 4 5 6]
```

- NumPy provides a helper function: vstack() to stack along columns:



```
import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

arr = np.vstack((arr1, arr2))

print(arr) >>>
[[1 2 3]
 [4 5 6]]
```

- Splitting is reverse operation of Joining. Joining merges multiple arrays into one and Splitting breaks one array into multiple. We use `array_split()` for splitting arrays, we pass it the array we want to split and the number of splits. mesela bir array'i üçe ayıralım:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr) >>> [array([1, 2]), array([3, 4]), array([5, 6])]
```

notice that the return value is an array containing three arrays.

- If the array has less elements than required, it will adjust from the end accordingly. arrayi 4'e ayıralım:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 4)

print(newarr) >>> [array([1, 2]), array([3, 4]), array([5]), array([6])]
```

- We also have the method `split()` available but it will not adjust the elements when elements are less in source array for splitting like in example above, `array_split()` worked properly but `split()` would fail. The return value of the `array_split()` method is an array containing each of the split as an array. If you split an array into 3 arrays, you can access them from the result just like any array element:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6])

newarr = np.array_split(arr, 3)

print(newarr[0]) >>> [1 2]
print(newarr[1]) >>> [3 4]
print(newarr[2]) >>> [5 6]
```

- `where()` metodu ile bir sayının hangi index(ler)de olduğunu görebilirsin:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 4, 4])

x = np.where(arr == 4)
```

```
print(x) >>> (array([3, 5, 6], dtype=int64),)
```

o yani 4 sayısı 3. 5. ve 6. indekslerde mevcutmuş  
çift olan sayıların indekslerini bulma:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
x = np.where(arr%2 == 0)
```

```
print(x)
```

- searchsorted() metodu performs a binary search in the array, and returns the index where the specified value would be inserted to maintain the search order. mesela let's find the indexes where the value 7 should be inserted:

```
import numpy as np
```

```
arr = np.array([6, 7, 8, 9])
```

```
x = np.searchsorted(arr, 7)
```

```
print(x) >>> 1
```

o The number 7 should be inserted on index 1 to remain the sort order.

- sort() metodu arrayi sıralar:

```
import numpy as np
```

```
arr = np.array([3, 2, 0, 1])
```

```
print(np.sort(arr)) >>> [0 1 2 3]
```

o however, bear in mind that this method returns a copy of the array, leaving the original array unchanged.

If you use the sort() method on a 2-D array, both arrays will be sorted.

- Random number does NOT mean a different number every time. Random means something that can not be predicted logically. Computers work on programs, and programs are definitive set of instructions. So it means there must be some algorithm to generate a random number as well. If there is a program to generate random number it can be predicted, thus it is not truly random. Random numbers generated through a generation algorithm are called pseudo random. Can we make truly random numbers? Yes. In order to generate a truly random number on our computers we need to get the random data from some outside source. This outside source is generally our keystrokes, mouse movements, data on network etc. fakat çoğu zaman pseudo random sayılar yeterli olacaktır, truly randomlar dijital rulet, şifreleme vs'de lazım oluyor.

- random sayılar üretmek için random modülünü import ediyoruz:

```
from numpy import random
```

```
x = random.randint(100)
```

```
print(x) >>> 47
```

- integer değil de 0-1 arası float üretmek için:

```
from numpy import random
```

```
x = random.rand()
```

```
print(x) >>> 0. 8005228465219864
```

rand(3) dersin 3 tane random float üretir

- random arrayler de oluşturabiliriz. mesela let's generate a 1-D array containing 5 random integers from 0 to 100:

```
from numpy import random
```

```
x=random.randint(100, size=(5))
```

```
print(x) >>> [81 12 80 62 40]
```

generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
from numpy import random
```

```
x = random.randint(100, size=(3, 5))
```

```
print(x) >>>
```

```
[[91 25 93 78 38]
```

```
 [30 56 42 71 99]
```

```
 [53 43 28 41 4]]
```

- Generate a 2-D array with 3 rows, each row containing 5 random numbers:

```
from numpy import random
```

```
x = random.rand(3, 5)
```

```
print(x)
```

- The choice() method allows you to generate a random value based on an array of values. The choice() method takes an array as a parameter and randomly returns one of the values:

```
from numpy import random
```

```
x = random.choice([3, 5, 7, 9])
```

```
print(x) >>> 4ünden birini random yazdırır
```

- buna size= metoduyla boyut da ekleyebilirsiniz:

```
from numpy import random
```

```
x = random.choice([3, 5, 7, 9], size=(3,3))
```

```
print(x) >>>
```

```
[[5 9 3]
```

```
 [3 5 9]
```

```
 [9 9 5]]
```

- Data Distribution is a list of all possible values, and how often each value occurs. Such lists are important when working with statistics and data science. The random module offer methods that returns randomly generated data distributions. A random distribution is a set of random numbers that follow a certain probability density function. Probability Density Function: A function that describes a continuous probability. i.e. probability of all values in an array. We can generate random numbers based on defined probabilities using the choice() method of the random module. The choice() method allows us to specify the probability for each value. The probability is set by a number between 0 and 1, where 0 means that the value will never occur and 1 means that the value will always occur.
- mesela let's generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9. The probability for the value to be 3 is set to be 0.1 The probability for the value to be 5 is set to be 0.3 The probability for the value to be 7 is set to be 0.6 The probability for the value to be 9 is set to be 0:

```
from numpy import random
```

```
x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], size=(100))
```

```
print(x) >>>
```

```
[7 5 7 3 5 7 7 7 7 3 5 7 3 3 5 7 7 3 7 7 7 7 5 7 7 5 7 7 3 5 3 7 5 5 7 7 7 7 5 7 7 7 5 7 3
7 7 7 7 5 3 7 7 5 7 7 5 7 7 5 5 7 7 7 3 5 7 7 3 5 7 7 7 7 7 7 7 3 5 7 7 7 7 5 5 7 7 7 7
5 5 7 5 3 5 5 5 7 7]
```

p'nin toplamı 1'e eşit olmak zorundadır

- random permutations: A permutation refers to an arrangement of elements. e.g. [3, 2, 1] is a permutation of [1, 2, 3] and vice-versa. The NumPy Random module provides two methods for this:

- o shuffle(): Shuffle means changing arrangement of elements in-place. i.e. in the array itself:

```
from numpy import random
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
random.shuffle(arr)
print(arr) >>> [2 4 3 1 5]
```

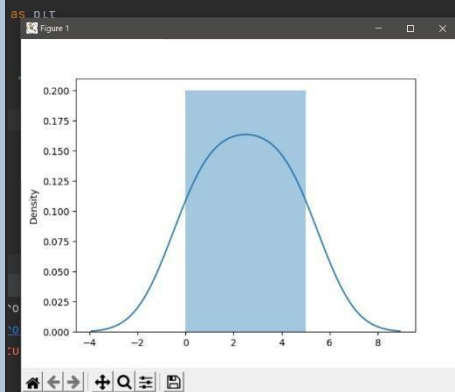
shuffle() method makes changes to the original array.

- o permutation(): The permutation() method *returns* a re-arranged array (and leaves the original array un-changed):

```
from numpy import random
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(random.permutation(arr))
```

- seaborn library: Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions. Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array. öncelikle bir görsel veri oluşturmak için matplotlib'i import ediyoruz, daha sonra seaborn'u. en son da verileri giriyoruz:

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5])
plt.show() >>>
```



arkadaki mavi histogram olmasın istiyorsan:

```
sns.distplot([0, 1, 2, 3, 4, 5], hist=False)
```

- The Normal Distribution is one of the most important distributions. It is also called the Gaussian Distribution after the German mathematician Carl Friedrich Gauss. It fits the probability distribution of many events, eg. IQ Scores, Heartbeat etc. Use the random.normal() method to get a Normal Data Distribution. It has three parameters:
  - o loc - (Mean) where the peak of the bell exists.
  - o scale - (Standard Deviation) how flat the graph distribution should be.
  - o size - The shape of the returned array.

örnek:

```
from numpy import random
x = random.normal(size=(2, 3))
print(x) >>>
[[-0.09062848 -1.53156663 -1.04175519]
 [-0.60475515 -0.37278037 1.59772858]]
```

generate a random normal distribution of size 2x3 with mean at 1 and standard deviation of 2:

```
from numpy import random
x = random.normal(loc=1, scale=2, size=(2, 3))
print(x)
```

- binomial distribution is a *Discrete Distribution* ( The distribution is defined at separate set of events, bağımsız olasılık) . It describes the outcome of binary scenarios, e.g. toss of a coin, it will either be head or tails. It has three parameters:
  - o n - number of trials
  - o p - probability of occurrence of each trial (ex. 0.5 for toss of a coin)
  - o size - shape of the returned array
- Given 10 trials for coin toss generate 10 data points:

```
from numpy import random
x = random.binomial(n=10, p=0.5, size=10)
print(x) >>> [6 6 5 6 6 8 4 3 6 4]
```

- let's visualize this:

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True, kde=False)

plt.show()
```

- The main difference is that normal distribution is continuous whereas binomial is discrete, but if there are enough data points it will be quite similar to normal distribution with certain loc and scale:

```
from numpy import random
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')
sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')

plt.show()
```

-

#pandas

- bir pandas veri dizisi oluşturalım:

```
import pandas as pd
import numpy as np
import random

data types
numbers = [10, 20, 33, 35, 39]
letters = ["x", "y", "z", "t"]
scalar = 5
dict = {'a':10, 'b':20, 'c': 30}
random_numbers = np.random.randint(10, 100, 5)

pandas_series = pd.Series(numbers)
pandas_series = pd.Series(letters)
pandas_series = pd.Series(scalar)
pandas_series = pd.Series(dict)
pandas_series = pd.Series(random_numbers)
```

numpy’da liste içeriği homojen yani aynı veri tipinde olmak zorundaydı fakat pandas’ta karışık olabilir.

- index bilgisini kendin de verebilirsin:

```
pandas_series = pd.Series(5, [0,1,2])
print(pandas_series) >>>
0 5
1 5
2 5
```

sayı yerine a, b, c... de verebilirsin  
elemanlara indeks yoluyla ulaşmak için:

```
pandas_series = pd.Series([1, 2, 3], ["birinci", "ikinci", "üçüncü"])
print(pandas_series[2]) >>> 3
```

pandas\_series["üçüncü"] de desen 3 gelirdi

- bir .csv’deki ilk 5 kaydı alma:

```
import pandas as pd
df = pd.read_csv("valorant.csv")
result = df.head(3)
print(result)
```

- toplam kaç kayıt vardır?

```
result = len(df.index)
```

- fire-rate sütununun ortalaması kaçtır?

```
result = df["fire-rate"].mean()
```

- en yüksek fire-rate (sütundaki en büyük sayı) kaçtır?

```
result = df["fire-rate"].max()
```

- en düşük fire-rate (sütundaki en küçük sayı) kaçtır?

```
result = df["fire-rate"].min()
```

- toplamı kaçtır?

```
result = df["fire-rate"].sum()
```

- değerleri toplamı:

```
result = pandas_series + pandas_series
```

- koşula tabi tutma:

```
result = pandas_series >= 50
```

50+ olanların karşısına True diğerlerinin False yazdırır

- dataFrames iki+ setin birleşimi gibi düşünülebilir. excel'e benzer.

| Series |   |  | Series  |   |  | DataFrame |         |   |
|--------|---|--|---------|---|--|-----------|---------|---|
| apples |   |  | oranges |   |  | apples    | oranges |   |
| 0      | 3 |  | 0       | 0 |  | 0         | 3       | 0 |
| 1      | 2 |  | 1       | 3 |  | 1         | 2       | 3 |
| 2      | 0 |  | 2       | 7 |  | 2         | 0       | 7 |
| 3      | 1 |  | 3       | 2 |  | 3         | 1       | 2 |

basit bir df (dataframe) örneği:

```
import pandas as pd
s1 = pd.Series([1, 2, 3, 4])
s2 = pd.Series([10, 20, 30, 40])

data = dict(apples = s1, oranges = s2)
df = pd.DataFrame(data)

print(df) >>>
 apples oranges
0 1 10
1 2 20
2 3 30
3 4 40
```

- data frame içeriğini manuel girelim:

```
data = [['Ahmet', 62], ['Ayşe', 50], ['Buğra', 35]]

df = pd.DataFrame(data, columns = ['isim', 'puan'], index=[1,2,3])

print(df) >>>
 isim puan
1 Ahmet 62
2 Ayşe 50
3 Buğra 35
```

- csv dosyasını okumak için:

```
df = pd.read_csv('valorant.csv')
print(df)
```

- json dosyasını okumak için:

```
df = pd.read_json('valorant.json', encoding="UTF-8")
print(df)
```

- excel dosyasını okumak için ayrıca library ihtiyacın vardır:

```
pip install xlrd
pip install openpyxl
df = pd.read_excel('valorant.xlsx')
print(df)
```

- mesela 3 sütundan oluşan bir kod yazalım:

```
import pandas as pd
from numpy.random import randn

df = pd.DataFrame(randn(3, 3), index=["A", "B", "C"], columns=["Column1", "Column2", "Column3"])
print(df) >>>
 Column1 Column2 Column3
A -0.370302 0.894559 1.074019
B 0.619447 -2.540169 -1.191083
C 0.141436 0.468215 0.832443
```

```
C 0.376498 0.423277 0.573816
```

- bu kodun birinci sütununu almak istiyosak:

```
result = df["Column1"]
print(result)
```

- 2+ sütununu almak istiyosak:

```
result = df[["Column1", "Column2"]]
print(result)
```

- birinci satırını almak istiyosak loc[ ] metodunu kullanmamız gerekir:

```
result = df.loc["A"]
```

- loc[ ] metodu şeması şöyledir:

```
loc["row", "column"]
```

önce satır sonra sütun belirtirsin. sadece kolon seçmek istiyosan

```
loc[:, "column"]
```

- belli aralıktaki satır-sütunları almak içinse:

```
result = df.loc[:, "Column2":"Column6"]
```

başlangıçtan itibaren alsın istiyosan da :

```
result = df.loc[:, "Column6"]
```

aynısını satırlarla da yapabilirsin. mesela A ve B satırlarını almak için:

```
result = df.loc["A":"B", "Column1"]
```

- yeni bir sütun oluşturmak için:

```
df["Column4"] = pd.Series(randn(3), ["A", "B", "C"])
```

- iki sütunun toplamını yazdırmak için:

```
df["ColumnToplam"] = df["Column1"] + df["Column2"]
```

- sütun silmek için drop() metodunu kullanıyoruz ve bir sütun olduğunu belirtmek için de axis=1 diyoruz:

```
df = df.drop("Column3", axis=1, inplace=True)
print(df) >>>
```

```
 Column1 Column2
A -0.701063 0.964256
B -1.044522 3.041006
C -1.262892 0.503822
```

default değeri false olan inplace'i True yapmazsan orijinal değişkende değişiklik yapmaz dolayısıyla şöyle kullanmalısın.

- mesela random sayılardan oluşan beş sütunlu bir data oluşturalım:

```
import pandas as pd
import numpy as np
data = np.random.randint(1, 100, 75).reshape(15,5)
df = pd.DataFrame(data, columns = ["Column1", "Column2", "Column3", "Column4", "Column5"])
print(df) >>>
```

```
 Column1 Column2 Column3 Column4 Column5
0 66 34 92 84 60
1 48 65 31 73 72
2 97 68 9 81 1
3 15 14 65 23 86
4 83 82 40 13 53
5 65 26 60 92 62
6 7 57 61 43 91
7 82 11 31 42 22
8 68 76 56 59 83
9 60 82 71 95 7
10 8 50 31 55 55
11 20 89 96 65 67
12 9 4 3 61 38
```



|    |    |    |    |    |    |
|----|----|----|----|----|----|
| 13 | 10 | 91 | 59 | 41 | 90 |
| 14 | 18 | 12 | 15 | 48 | 33 |

- sütunların isimlerini öğrenmek için:

```
result = df.columns
```

- head() metodu ile ilk 5 tail() metodu ile son 5 kaydı alabilirsin. ya da kaç tane istiyosan yazıyosun:

```
result = df.tail()
```

```
result = df.head()
```

- belli bir sütunun ilk 5 kaydını almak için:

```
result = df["Column4"].head()
```

- dataframe'de parçalama yapmak için slicing kullanabilirsin. böylelikle mesela head() metodu ile ilk 5 kaydı değil de df[10:] diyerek 10. kayıttan sonraki ilk 5 kaydı almış olursun

```
result = df[10:].head()
```

- verisi 50'den büyük olanlar için True diğerleri için False yazdıran kod:

```
result = df > 50
```

```
print(result)
```

gösterebilirsin istiyosan da:

```
result = df[df > 50]
```

```
print(result)
```

bunda başka matematiksel operatörler (df % 2 == 0 vs.) de kullanabilirsin

- NaN → Not a Number

- şimdi bunları bi örnekte kullanalım. mesela imdb top 250 listesinin olduğu imdb250.csv isimli bir dosyamız olsun. bu dosyada şunları yapalım:

- öncelikle dosyayı okuyalım:

```
import pandas as pd
```

```
import numpy as np
```

```
df = pd.read_csv("imdb250.csv")
```

- ilk 5 kaydı alalım:

```
result = df.head()
```

head()'in içine ne yazarsan o kadar ilk sıradan kayıt getirir

- son 5 kaydı alalım:

```
result = df.tail()
```

- sadece yıl sütununun ilk 5 kaydını alalım:

```
result = df["IMDBYear"].head()
```

- birden fazla sütunun son 3 kaydını alma:

```
df[["IMDBYear", "Cast3"]].tail(3)
```

birden fazla sütun seçeceksen bir kere daha [ ] içine alman gerekir.

- bir sütundan ilk değil de ikinci beş kaydı alalım:

```
result = df[5:][["IMDBYear", "Cast3"]].tail(3)
```

- puanı 8+ olanlardan ilk 50 tanesi alma:

```
result = df[df["Rating"] ≥ 8.0][["Date", "Title", "Rating"]].head(20)
```

- 1985-1990 arası filmleri alma:

```
result = df[(df["Date"] ≥ 1985) & (df["Date"] ≤ 1990)][["Title", "Date"]].head(15)
```

- df.["Sütunİsmi"].sum() metodu ile sayısal bir sütundaki verileri toplayabilirsin:

```
import pandas as pd
```

```
import numpy as np
```

```
df = pd.read_csv("imdb250.csv")
```

```
result = df["Rating"].sum()
```

```
print(result)
```

- groupby()

- mean()

- min()
- max()
- agg()
- count()
- şu kod sana NaN'lı satırlar verir:

```
import pandas as pd
import numpy as np

data = np.random.randint(10, 100, 15).reshape(5,3)
df = pd.DataFrame(data, index=["a", "c", "e", "f", "h"], columns=["sütun1",
"sütun2", "sütun3"])

df = df.reindex(["a", "b", "c", "d", "e", "f", "g", "h"])

result = df
print(result)
```

çünkü df'yi tanımlarken b d ve g yazmamıştık.

- nan'lı satırları silmek için:

```
import pandas as pd

data = pd.read_csv('valorant.csv')

data.dropna(inplace=True)
```

ya da

```
data = data.dropna()
```

- bir csv dosyasında bir sütundaki elemanları büyük harfe çevirme:

```
import pandas as pd

data = pd.read_csv('valorant.csv')

data['Name'] = data['Name'].str.upper()
```

burada data değişkenine atadığımız csv dosyasının Name sütunun altındaki str'leri BÜYÜK harfe çevirdik. aynısını lower() da yapabiliriz.

- contains metodu ile bir sütunda belli bir anahtar kelimeyi ararız:

```
import pandas as pd

data = pd.read_csv('valorant.csv')

data = data[data["Weapon Type"].str.contains("Rifle")]

print(data) >>> sadece Weapon Type sütununda Rifle yazanları yazdıracaktır
```

- bir satır ya da sütunu silmek için drop() metodunu kullanırız:

```
result = df.drop("sütun1", axis= 1)
print(result)
```

axis= 0 dersen satıra 1 dersen sütuna karşılık gelir. fakat burada columns= değişkenindeki bir elemanı (sütunu) yazdığın için 0 diyemezsin hata verir.

birden fazla sütun silmek istiyosan:

```
result = df.drop(["sütun1", "sütun2"], axis= 1)
print(result)
```

aynı şekilde a satırını silmek istiyosan da:

```
result = df.drop("a", axis=0)
print(result)
```

birden çok satırı silmek istiyosan da:

```
result = df.drop(["a","b","c"], axis=0)
print(result)
```

- peki bu NaN (not a number) olan bozuk satırları nasıl tespit edeceğiz? bunun için isnull() metodunu kullanıyoruz:

```
result = df.isnull()
print(result) >>>
```

|   | sütun1 | sütun2 | sütun3 |
|---|--------|--------|--------|
| a | False  | False  | False  |
| b | True   | True   | True   |
| c | False  | False  | False  |
| d | True   | True   | True   |
| e | False  | False  | False  |
| f | False  | False  | False  |
| g | True   | True   | True   |
| h | False  | False  | False  |

- NaN olmayanları almak için de notnull() metodunu kullanıyoruz.
- her satırda kaç adet NaN olduğunu bulmak için isnull() + sum() metotlarını kullanıyoruz:

```
result = df.isnull().sum()
print(result)
```

- sadece belirli bir sütundaki NaN değerlerini bulmak için ise:

```
result = df["sütun3"].isnull().sum()
print(result)
```

- 
-

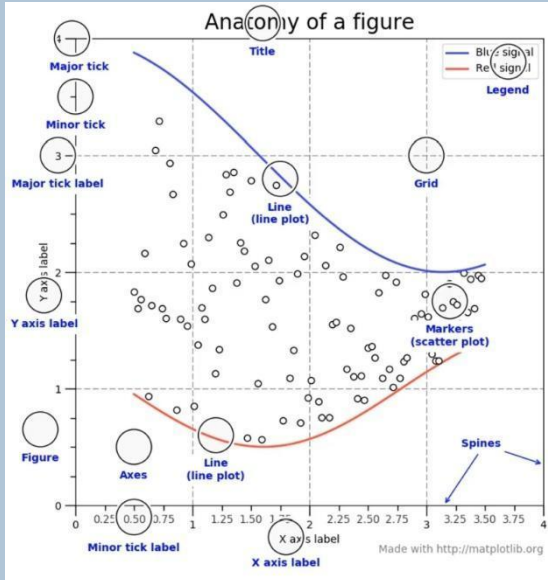
## #matplotlib

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. python'da pip yüklü ise terminalde `pip install matplotlib` diyerek yüküyoruz. daha sonra `import matplotlib` diyerek kullanıma hazır hale getiriyoruz.

- version sorma:

```
import matplotlib
print(matplotlib.__version__)
```

- matplotlib şeması:

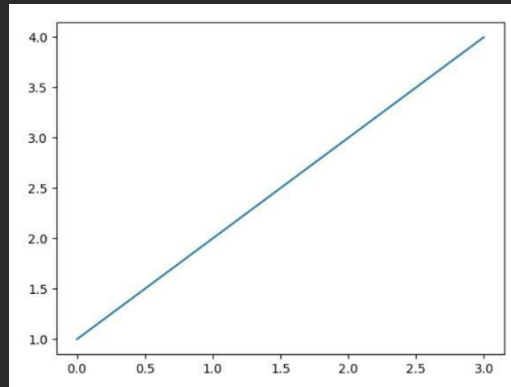


- basit bir matplotlib grafiği yazalım:

```
import matplotlib.pyplot as plt
import numpy as np

x = [1,2,3,4]

plt.plot(x)
plt.show() >>>
```



- x ve y'li bir grafik yazalım:

```
import matplotlib.pyplot as plt
import numpy as np

x = [1,2,3,4]
y = [1, 4, 9, 16]

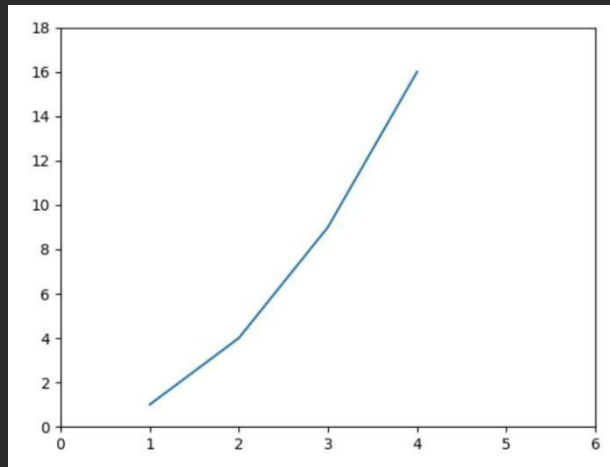
plt.plot(x,y)
plt.show()
```

- axis ve ordinat doğrusundaki rakamları kendimiz de belirleyebiliriz:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = [1,2,3,4]
y = [1, 4, 9, 16]

plt.plot(x,y)
plt.axis([0,6, 0, 18])
plt.show() >>>
```



- grafiği özelleştirmek istersek:

- o plt.title() metodu ile grafiğe isim verebilirsin
- o plt.xlabel() metodu ile x eksenine isim verebilirsin
- o plt.ylabel() metodu ile y eksenine isim verebilirsin
- o plt.plot(x, y, color="red") metodu ile rengini değiştirebilirsin
- o plt.plot(x, y, linewidth = 5) metodu ile kalınlığını değiştirebilirsin
- o plt.plot(x, y, "--") metodu ile kesik çizgi yapabilirsin
- o değdiği noktaları işaretlemek için circle marker kullanabilirsin:

```
plt.plot(x, y, "ob--")
```

- o burada circle'ı, -- kesik çizgililiğini b blue'yu indike eder.

matplotlib plot style aratarak fazlasına ulaşabilirsin

- birden fazla çizgiyi grafiğe ekleme:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)

plt.plot(x, x, label = "linear")
plt.plot(x, x**2, label = "quadratic")
plt.plot(x, x**3, label = "cubic")

plt.xlabel("this is x line")
plt.ylabel("this is y line")

plt.legend()
plt.show()
```

- eğer çizgiler farklı grafiklerde olsun istiyosan:

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100)
fig, axs = plt.subplots(3)

axs[0].plot(x, x, color="red")
```

```
axs[1].plot(x, x**2, color="blue")
axs[2].plot(x, x**5, color="black")
plt.show()
```

bunlara başlık vermek istiyosan:

```
axs[0].plot(x, x, color="red")
axs[0].set_title("Linear")
```

```
axs[1].plot(x, x**2, color="blue")
axs[1].set_title("quadratic")
```

```
axs[2].plot(x, x**5, color="black")
axs[2].set_title("cubic")
```

```
yazılar birbirine karışmasın diye görüntüyü ayarlar
plt.tight_layout()
```

- bu grafikleri alt alta değil de yan yana dizmek istiyosan:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(0, 2, 100)
fig, axs = plt.subplots(2,2)
fig.suptitle = ("graph başlık")
```

```
axs[0,0].plot(x, x, color="red")
axs[0,1].plot(x, x**2, color="blue")
axs[1,0].plot(x, x**3, color="green")
axs[1,1].plot(x, x**4, color="black")
```

```
plt.show()
```

- dörtlü boş grafik oluşturmak için:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(-10, 9, 20)
y = x ** 3
```

```
fig, axs = plt.subplots(2, 2)
```

```
plt.show()
```

- örnek:

```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(-10, 9, 20)
y = x ** 3
z = x ** 2
```

```
figure = plt.figure()
```

```
figür üzerine eklenilecek olan axes alanının konumunu ayarlıyoruz
soldan ne kadar, sağdan ne kadar uzak, genişliği, yüksekliği belirler
axes = figure.add_axes([0.1, 0.1, 0.8, 0.8])
```

```
x ve y değerlerinin çizgi grafiğini ekliyoruz ve b(lue) rengini seçiyoruz
```

```

axes.plot(x, y, "b")

axes.set_xlabel("x axis")
axes.set_ylabel("y axis")
axes.set_title("main graph")

grafiğin içine grafik eklemek için
axes2 = figure.add_axes([0.15, 0.6, 0.25, 0.25])
axes2.plot(x, z, "r")
axes2.set_xlabel("x axis")
axes2.set_ylabel("y axis")
axes2.set_title("2. graph")

legend(loc=1,2,3,4) diyerek konumunu değiştirebilirsin
plt.legend()
plt.show()

```

- farklı tür grafikler:
  - o alan grafiği aka stack plot
  - o pasta grafiği aka pie plot
  - o bar grafiği

alan grafiği (stack plot):

```

import matplotlib.pyplot as plt

yıl = [2011, 2012, 2013, 2014, 2015]

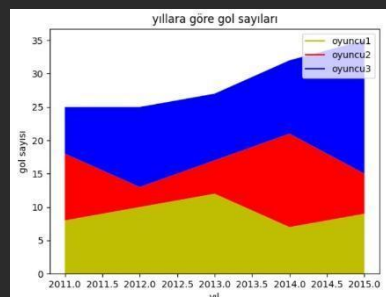
oyuncu1 = [8, 10, 12, 7, 9]
oyuncu2 = [7, 12, 10, 11, 20]
oyuncu3 = [10, 3, 5, 14, 6]

plt.plot([], [], color = 'y', label= "oyuncu1")
plt.plot([], [], color = 'r', label= "oyuncu2")
plt.plot([], [], color = 'b', label= "oyuncu3")

plt.stackplot(yıl, oyuncu1, oyuncu3, oyuncu2, colors=["y","r","b"])

plt.title("yıllara göre gol sayıları")
plt.xlabel("yıl")
plt.ylabel("gol sayısı")
plt.legend ()
plt.show() >>>

```



pie plot:

```

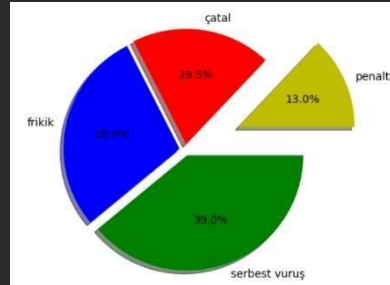
import matplotlib.pyplot as plt

goal_types = "penaltı", "çatal", "frikik", "serbest vuruş"

```

```
goals = [10, 15, 22, 30]
colors = ["y", "r", "b", "g"]
```

```
shadow ile gölge eklersin explode ile birbirinden ayırırsın dilimleri ve
pastadan uzakta gösterilsin istiyosan onun explode'unu büyültebilirsin
yüzdeler gösterilsin istiyosan "%1.1f%%" formülünü kullanmalısın autopct metoduyla
plt.pie(goals, labels=goal_types, colors=colors, shadow=True, explode=(0.5, 0.05, 0.05, 0.05), autopct="%1.1f%%")
plt.show() >>>
```

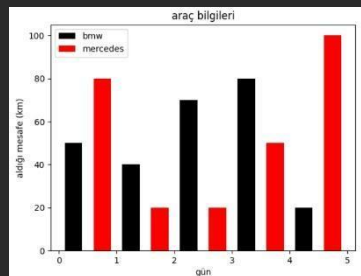


bar grafiği:

```
import matplotlib.pyplot as plt

iki veriden birincisi gün bilgisi 2.si o güne aldığı mesafe
plt.bar([0.25, 1.25, 2.25, 3.25, 4.25], [50, 40, 70, 80, 20], label = "bmw",
width=0.3,color="black")
plt.bar([0.75, 1.75, 2.75, 3.75, 4.75], [80,20,20,50, 100], label = "mercedes", width=0.3,
color="r")

plt.legend(loc=2)
plt.xlabel("gün")
plt.ylabel("aldığı mesafe (km)")
plt.title("araç bilgileri")
plt.show() >>>
```



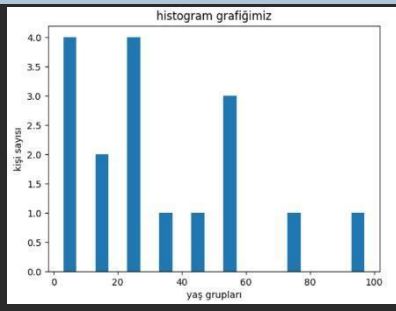
histogram grafiği:

```
import matplotlib.pyplot as plt

yaşlar = [22,55,23,6,57,45,23,98,4,7,35,76,8,24,10, 18, 50]
yaş_grupları = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.hist(yaşlar, yaş_grupları, histtype="bar",rwidth=0.4)
plt.xlabel("yaş grupları")
plt.ylabel("kişi sayısı")
plt.title("histogram grafiğimiz")
plt.show() >>>
```





## #BeautifulSoup

- prettify() metodu dağınık yazdığın html kodunu düzenler:

```
html_doc = """
<!DOCTYPE html>
<html lang="en">
 <head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial- scale=1.0">
 <title>Document</title>
 </head>
<body>
 merhaba
</body>
<h1>
 asdasd
</h1>
<div>
 asdas
</div>
</html>
"""
```

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, "html.parser")
result = soup.prettify()
print(result)
```

- belirli bir bilgiyi almak için:

```
result = soup.title
```

ya da

```
result = soup.body
```

-

#django

- bilgisayarında yüklü olan pip paketleri ve sürümlerini öğrenmek için pip freeze yaz terminale
- terminale django-admin help yazarak hangi komutla proje taslağı oluşturabileceğinin listesini görebilirsin.daha sonra bu isimlerden birini django-admin startproject projeİsmi yazarak taslak oluşturabilirsin.

#selenium

- selenium bir web otomasyon kütüphanesidir. websitelerini ziyaret edip etkileşimde bulunabiliriz. things you can do with selenium:
  - o drag and drop
  - o click
  - o scroll
  - o open & close a webpage
  - o fill in forms
- hangi tarayıcıda çalışacaksak o tarayıcının selenium driverının .py dosya konumunda olması gerekiyor.:

```
from selenium import webdriver
mozilla selenium driverı dosyaKonumuna attık
driver isimli değişkene webdriver.istediğimizTarayıcı()'yı atıyoruz
driver = webdriver.Firefox()

istediğimiz sitenin url girdik
url = "https://techwithtim.net"

get() metodunu kullanıyoruz
driver.get(url)

sekmenin adını yazdırır
print(driver.title)

sekmeyi kapatmak için close() kullanıyoruz
driver.close()

tarayıcıyı kapatmak için quit
driver.quit()
```

- maximize.window() metodu ekranı büyütür
- save\_screenshot("ss\_ismi.jpg") o anki ekranın bu isimde bir ekran görüntüsünü alır:
- eğer farklı bir sayfaya yönlendirmek istiyorsan:

```
url = "http://yeniSayfa.com"
driver.get(url)
```

yani yeni bir url tanımlayıp get() metoduyla o url'ye gidiyorsun

- back() metoduyla bir önceki sayfaya dönersin:

```
driver.back()
```

forward() metoduyla da geriye gittikten sonra bir sonraki sayfaya gidebilirsin:

```
driver.forward()
```

- to click something you say:

```
element.click()
```

- bir kutucuğun içeriğini silmek için:

```
element.clear()
```

- bir sitedeki arama kutucuğunu kullanmak için önce o search bara inspect diyip ismini öğreniyoruz (xpath de kullanılabilir):

```
<div class="p42AE g311 jsname="vut3w 7X7div">
<input class="glFyf gsfi" jsaction="paste:puy29d;" maxlength="2048"
name="q" type="text" aria-autocomplete="both" aria-haspopup="false"
autocapitalize="none" autocomplete="off" autocorrect="off"
autofocus="" role="combobox" spellcheck="false" title="Ara" value=""
aria-label="Ara" data-ved="0ahUKEwiT-
7S4yfLyAhUoyzgGHQxxA5YQ39UDCAQ"> flex
```

ismi "q" miş. şimdi kodda bunu kullanalım:

```

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

driver = webdriver.Firefox()
url = "https://google.com"
driver.get(url)

searchbar'a inspect edip ismini öğreniyoruz
search = driver.find_element_by_name("q")

"tutorials" arasın diye send_keys() kullanıyoruz
search.send_keys("how to learn a language")

arasın diye enter (aka return) type ettiriyoruz
search.send_keys(Keys.RETURN)

```

- şimdi arama yaptıktan sonra bir linke tıklayalım:

```

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

driver = webdriver.Firefox()
url = "https://google.com"
driver.get(url)

driver.maximize_window()
time.sleep(1)

bu metotla ekranda ismen arama yapıp tıklıyoruz
link = driver.find_element_by_link_text("Görseller")
link.click()

```

fakat burada şöyle bir sorun var, click komutu vermeden önce sayfanın yüklenip o ismin ekran gözüküyor olduğundan emin olmamız gerek. aksi taktirde hata verecektir. bunu bekleyerek garantiye almak için try finally metodunu kullanıyoruz:

```

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

driver = webdriver.Firefox()
url = "https://google.com"
driver.get(url)

driver.maximize_window()
time.sleep(1)

bu metotla ekranda ismen arama yapıp tıklıyoruz
link = driver.find_element_by_link_text("Görseller")
link.click()

wait max 10 seconds
try:

```

```
element = WebDriverWait(driver, 10).until(
 EC.presence_of_element_located((By.LINK_TEXT, ("Görseller"))))
)
element.click()
except:
 driver.quit()
```

şimdi bir arama yapıp entera tıklayalım:

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import time

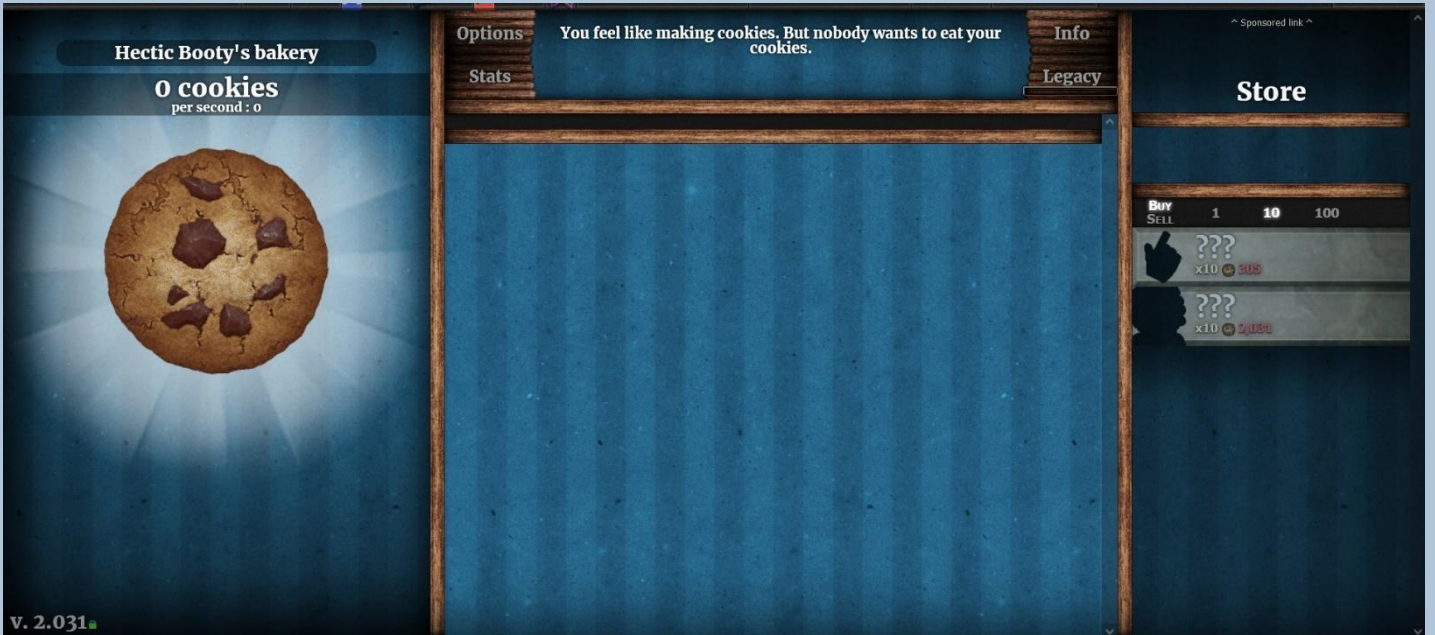
driver = webdriver.Firefox()
url = "https://google.com"
driver.get(url)

driver.maximize_window()
time.sleep(1)

bu metotla ekranda ismen arama yapıp tıklıyoruz
link = driver.find_element_by_link_text("Görseller")
link.click()

wait max 10 seconds
try:
 search = driver.find_element_by_name("q")
 search.send_keys("kıızılgırdan")
 search.send_keys(Keys.ENTER)
except:
 driver.quit()
```

- fareyle tıklama, basılı tutma vs. gibi aksiyonları yapan program bi program yazalım. bunun için cookie clicker isimli bir oyunu kullanacağız:



kurabiye resminin üzerine tıkladıkça yukarıdaki sıfır 1 artıyor ve store'dan bir şeyler satın almamızı sağlıyor. otomatik tıklayan program yazalım ve mesela 20 kere tıklama işlemi yaptıralım:

```
from selenium import webdriver
from selenium.webdriver import ActionChains
import time

driver = webdriver.Firefox()
url = "https://orteil.dashnet.org/cookieclicker/"
driver.get(url)

sayfayı büyütüyor
driver.maximize_window()
time.sleep(1)

biraz bekliyoruz ki ekran yüklensin
driver.implicitly_wait(4)

kurabiyenin id
cookie = driver.find_element_by_id("bigCookie")

kurabiye sayacının id
cookie_count = driver.find_element_by_id("cookies")

"""upgradelerin fiyat bilgisi productPrice0,1,2 diye gidiyor. bunları sırasıyla
bir listede toplamak için range metodunu kullanıyoruz. 0, 1 değil de
-1 diyerek 1, 0 (tersten) yazdırmamızın sebebi paramız varsa (click sayısı) en pahalı
olanı alsın, ucuz olandan birçok kere satın almasın diye"""
items = [driver.find_element_by_id("productPrice" + str(i)) for i in range(1,-1,-1)]

click işlemi tanımlıyoruz
actions = ActionChains(driver)
actions.click(cookie)

range ile sayısını belirliyoruz
for i in range(100):
 # perform() demezsek eylemler işleme sokulmaz
 actions.perform()

 # tıklanılan sayacı yazdırır
 count = int(cookie_count.text.split(" ")[0])
 print(count)

 # upgrade satın almamızı sağlar
 for item in items:
 value = int(item.text)
 if value ≤ count:
 upgrade_actions = ActionChains(driver)
 upgrade_actions.move_to_element(item)
 upgrade_actions.click()
 upgrade_actions.perform()
```

#SQL

- ubuntuya mysql'i şu şekilde kurabilirsin  
<https://phoenixnap.com/kb/install-mysql-ubuntu-20-04>  
sudo systemctl status mysql diyerek de serverın durumunu kontrol edebilirsin
- veri yapıları genel olarak 2ye ayrılır
  - o SQL. bunun örnekleri
    - mySql
    - msSql
    - SQLite
  - o noSQL
    - mongodb

başka sınıflandırma olarak server-tabanlı, server-tabanlı olmayan olarak da ayrılabilir

- o mySql +
- o msSql +
- o mongodb +
- o sqlite (bu server tabanlı değil)

yani sqlite harici diğerleri internette depolar, sqlite yerel bilgisayarda bir json'da depolar verileri.

- üç tür ilişki tipi vardır
  - o one to one
  - o one to many
  - o many to many

1) one to many:

Products				Category	
id	name	price	catid	id	name
1	Samsung S6	2000	1	1	Telefon
2	Samsung S7	3000	1	2	Bilgisayar
3	Dell Laptop	5000	2	3	Tablet

burada products tablosunun her bir elemanı, category tablosundan sadece bir tanesinin kodunu alabilir (üçünden biridir). fakat category tablosunun her bir elemanı products'ta birden fazla elemana yazılabilir. buna one to many ilişki denir.

2) one to one

Products			Product_Details			
id	name	price	id	renk	ebat	agirlik
1	Samsung S6	2000	1	kırmızı	100x50	150gr
2	Samsung S7	3000	2	mavi	100x50	150gr
3	Dell Laptop	5000	3	siyah	100x50	3500gr

productstaki her bir eleman product\_detail'deki yalnızca tek bir elemanla ilişkilendirilebiliyorsa one-to-one bir ilişki vardır bu veriyapısında.

3) many to many



Products

id	name	price
1	Samsung S6	2000
2	Samsung S7	3000
3	Dell Laptop	5000

Category

id	name
1	Telefon
2	Bilgisayar
3	Elektronik

ProductCategory

productid	categoryid
1	1
1	3
2	1

yukarıda 1 numaralı productId'ye sahip eleman (s6) hem 1 (telefon) hem 3 (elektronik) kategorilerine dahil edilmiş. bu durumda ortada many to many bir ilişki vardır.

- 
- veri tabanında
  - o create
  - o read
  - o update
  - o delete

gibi işlemler yapabilirsin

- mySql'den mySql server ve mySql Workbench indiriyoruz 3