Python

* python is from the C family
* python can run without dependencies and is cross-platform. Meaning it can run on windows-Linux or android (ARM CPU) and ios ( ARM cpu)
* By default, python is installed on Linux family operating systems
* In Python, the source is compiled into a much simpler form called bytecode
* It can be executed line by line
* In .Net C# for example we need some dependencies to run prog.exe (.Net framework), but in python, we don't need any dependencies.

## books:

* python oreily
* Learn python the hard way
* Automating the boring stuff
* Nostratch
* Think like a programmer
* realpython.com
* automating boring stuff
* <https://allitbooks.net/> // all IT computer science books

## Sites:

**python learn**

* python.org
* codecademy python
* pythonforbiginners.com
* [Python Tutorial for Absolute Beginners](https://www.youtube.com/watch?v=Z1Yd7upQsXY)

**regex**

* jadi GitHub, jadi.net, jadi regex @radio.geek

**courses**

* Coursera
* edx
* GoToClass // for Arjang institute, it's online classes
* Vahid jamilo // to configure the server, the name is not correct
* <https://sokanacademy.com/>
* <https://code.tutsplus.com/> // to learn to code
* coderprog.com //books and video courses
* git.ir -> // good courses. Apache Cordova: mobile programming framework with html and css

**modules and codes**

* pypi.org //install module, follow comment. Mysql
* web interaction with python
* Tutorialspoint.com/http // to learn HTTP
* <https://realpython.com/> // to learn modules like beautifulsoup
* [Better Python dependency while packaging your project](https://medium.com/python-pandemonium/better-python-dependency-and-package-management-b5d8ea29dff1)
* About Project Euler // Project Euler is a series of challenging mathematical/computer programming problems that will require more than just mathematical insights to solve.
* Github // object detection

**clouds**

* Arvancloud.com, <https://www.fandogh.cloud/>, <https://backtory.com/> ( if the Django website is in one place and the DB in another place, to request remote db), firebase(google).
* Sokhanak.com
* Check-host.net // Check server: Check host - online website monitoring

**Security**

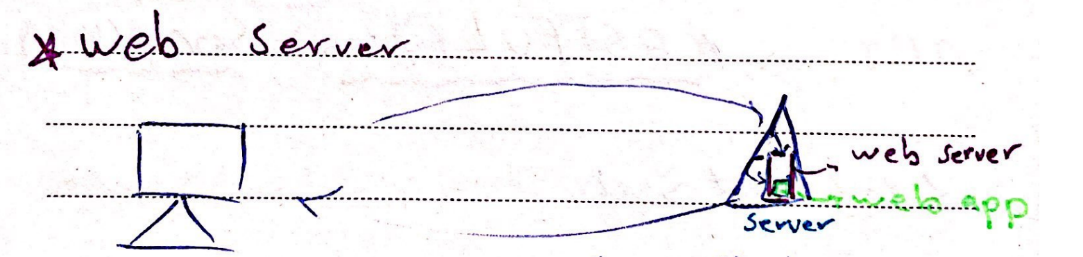
* [Google CTF: Beginner Quest: MEDIA-DB (SQL Injection)\_John Hammond](https://www.youtube.com/watch?v=qQgZQx6j9gg) // security
* Mermoryleaks.ir //security
* inform bugs // Fortweb(web app firewall), Hackerone, bug crowd
* first.org/cuss // score bugs

**Youtube:**

* Python data science
* AI with python
* Tech with Tim
* Tech With Tim // good python modules and tips
* john hammond // security

## Terminology:

* IDE // integrated Development environment. Python environment is called this. Python codes are interpreted line by line, **Python is not a compiler language**. So to translate it into machine language there should be no errors.
* in Windows: **SEH**=> Structured Exception Handling. is an exception handling mechanism included in most programs to make them robust and reliable. It is used to handle many types of errors and any exceptions that arise during the normal execution of an application.
* **PEP** => Python Enhancement Proposal, python standard. To know naming rules you should check the standard.
  + Def => say\_my\_name
  + Var => my\_name // no number for the first letter. The first letter can be (a-z, A-Z \_)
  + names: no number at first(a-z,A-Z,\_)(a-z,A-Z,\_,0-9)\*
* **cuss** => score bugs // what level the bug is, how serious it is. <https://www.first.org/cvss/training>
* **owasp** (open web application security project) => app test standard
* Lorem Ipsum
* Html // hypertext markup language
* **HTTP** // hypertext transfer protocol, **request/ response**
* one time password, OTP // ramze dovom
* **web Application** // php, ruby on rails, Django, flask, …
* **web service** // works with **URL** and **HTTPS**. Example: vajeyab.com
* **webserver** // iis, **Apache**, **Tomcat**, nginx, lightpd, Litespeed. **It's an API**.
* **server or host** // windows, Linux (ubuntu, mint, Debian, …)



* **PWA** (progressive web apps)
* **SDK** // software development kit, library, module, plugin
* **DevOps** // development and operations. DevOps is the **combination of cultural philosophies, practices, and tools** that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes.
* Clr .net java // are **memory safe**
* csv format: Derry, Rose, Maryam // data minige usage

Family name, Given name, Id

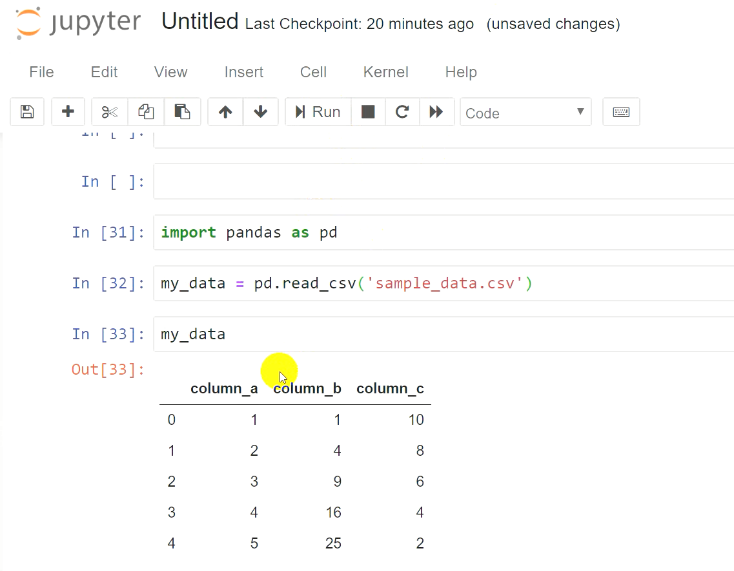
Derry, JJ, 232

Rose, pye, 230

## Python usage:

* hacking WAF (web Application Firewall)
* Desktop, web development, AI
* Data mining, Bitcoin, and the Web interact
* Image processing, Game
* Send sms
* Call
* Download
* Security
* Network
* Python for penetration tester (SEC573)
* easy face detection

## Tools:

* git bash // a Linux base shell
* process explorer // a more advanced alternative tool to task manager
* VI editor // for c#
* Source code pro // semi bold, fonts for editors
* db browser windows // a tool, DB Browser for SQLite (DB4S) is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite.
* chrome user agent switcher // we can change the chrome version
* ipython // interactive computing. Interactive shells
* The Jupyter Notebook // in cmd: **ipython notebook**. It's a tool for the web browser. It is an interactive computational environment. It feels like coding weblogs. It is included in **Anaconda**.
  + Shift + enter
  + Markdown language: backquote makes a code block for example.
  + Markdown cheatsheet
* Anaconda // has ipython inside it. Anaconda is **a distribution of the Python and R** programming languages for scientific computing, that aims to simplify **package management and deployment**. The distribution includes data-science packages suitable for Windows, Linux, and macOS.
* Mangodb // document-oriented, cross-platform
* Apache CouchDB // is an open-source document-oriented NoSQL database

## Python scripts and tools

* \Tools and \Scripts // add to Environment Variables

Editor Tips

## Editors

* Idle (Integrated Development Iand Learning Environment) => python editor
* sublime
* pycharm

## Mixed shortcuts

* on that object
* Notpad++: **ctr+d** // copies that line to the next line
* Notepad ++: **shift+tab**. Goes to the beginning of the line.
* python shell: **ctrl+ n** // new-file
* JetBrains: ctr+w -> one sentence-> another time-> all
* Notpad++ and sublime(not shell):word wrap // in view tab
* **Wordmap**, document map
* In win explorer: **Shift + right** click // to open the shell wherever we are
* notepad++ // language tab to save with a specific format
* in cmd: echo "Hi" **>** file.text // replace the content
* in cmd: echo "Hi" **>>** file.text // add (append) to the content
* in sublime: for + tab , def + tab
* in idle: x=- // if we forgot to assign the value, I don't know if the syntax it's correct. check

## pycharm tips:

* ctr + click => go to the source
* alt + back => go back
* add config => python.exe // to run python in pycharm we should set the python interpreter address and then in the run tab we can run files
* material them for pycharm
* in view tab => presentation mode
* .idea // it has the configured data, so when you want to give data to someone, you can delete it.
* Setting => wheel // to activate zoom and change font-size
* Alt + enter // corrects error
* Alt + enter // Ignore error like this
* Ctrl + alt + L // Prettify => reformat
* Refactor => rename // in all files
* Scroll => close file

## Debugging with pycharm:

* In debug section
* break point => program is executed to that point
* step over => next line
* step into=> into function
* on the fly, =>change the variable value after the assignment

Terminal

* **brew install python** // mac terminal
* **apt-get install python** // Linux
* **python -m pip install –upgrade pip** // one way to install pip
* using pip be careful what you install, the names are important
* pip install module\_name or pip2... or pip3 ... // if pip didn't work use pip 2 or 3
  + pip install mysql-connector-python
  + pip install pyQt5
  + pip install pyqt5-tools
  + pip install **B**eautiful**S**oup4
  + pip install requests
  + pip install names // dir(names)
* sqlite3, os, json, time, re // are included in the standard library
* pip3 –version
* help(i.getText) // this I don't know what it is
* ipython // After installing ipython you can run it. With **ctrl + d** you can exit.

syntax and escape chars

* Python is **case-sensitive** ( PHP is not )
* Indentation: **4 spaces**

## Operands

* >=
* > // When you compare string and number, if the word is one letter it returns an ASCII code.
* **\*\*** // power
* **not > and > or** // priority in logical statements
* 4.5 **//** 2 // rounds to floor

## Carriage chars

* **CRLF** // **carriage return line feed**. end of the line. C: \r LF: \n

## Escape Characters

* Scape means a **\** is behind the char
* **\n** // Linux
* **\r** // windows
* **\r \n** //mac
* So use **\r \n** to cover all
* **\b** //backspace
* **\a** // beep sound
* ' "" ' or " ' ' " or " \"\" "=>One should be single and the other double quote
* " "" ",' '' '=> This will be interpreted as two separated strings

## Variable scope

* **global** var name

## errors

* **Off by one** => in loops. An off-by-one error or off-by-one bug is a logic error involving the discrete equivalent of a boundary condition. It often occurs in computer programming when an iterative loop iterates one time too many or too few.
* **Business logical flaw (vulnerabilities)** // divide to 0

stock = stock – ( a negative stock) // it adds to it but should reduce from it

* **PHP type juggling** // PHP type juggling vulnerabilities arise when **loose comparison** (== or !=) is employed instead of **strict comparison** (=== or !==) in an area where the attacker can control one of the variables being compared.
* Boolean returns **true** if the variable is a string so be careful

General Syntax and funtions:

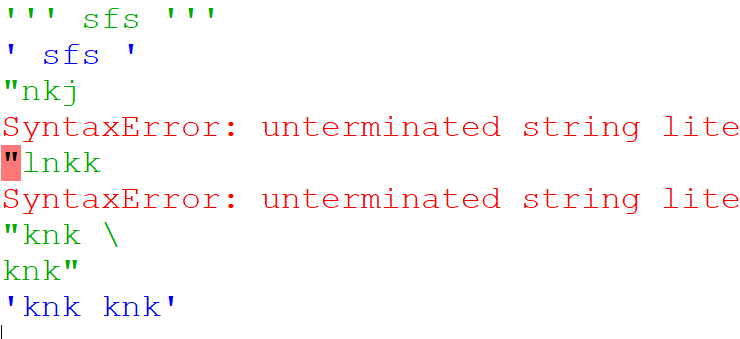
* **1j** // complex numbers
* **name1, name2 = 'Derry', 'Rose'**

## Comments

* **#** // Single Comment
* **#!** // Windows can't run this by default, it's for Unix/Linux. It's not a python thing, it a hashbang (or shebang) line that demonstrates which translator should handle the document. The Shebang #! symbol indicates the interpreter, or which version of an interpreter, to use when executing a script. If you include the code below in the first line of a file, it will use that interpreter for that file. It is like writing the command: /usr/local/bin/python3 xyz in which xyz is the name of the file The code below chooses the interpreter binary address for the file xyz.

**#! /usr/bin/python**

* **''' '''** // A kind of **string litterals**. It's not python standard but can be used. A string literal (with ' ' or " ") can span multiple lines, but there must be a backslash \ at the end of each line to escape the newline. String literals inside triple quotes, """ or ''', can span multiple lines of text. Python strings are "immutable" which means they cannot be changed after they are created (Java strings also use this immutable style). Since strings can't be changed, we construct \*new\* strings as we go to represent computed values. So for example the expression ('hello' + 'there') takes in the 2 strings 'hello' and 'there' and builds a new string 'hellothere'.



## Methods

**Dunder Methods**

* **\_\_nameOfMethod\_\_** // **dunder** (magic) methods, In Python, methods with double underscores (\_\_) are special methods, also known as dunder (double underscore) methods or magic methods. These methods provide functionality that can be **customized or overridden** for specific classes. Here are some commonly used dunder methods in Python: They are not usually used directly. The point of the "magic methods" is to let you customise how your object behaves when used with normal syntax. Sure, you could rewrite every foo = bar + baz as foo = bar.\_\_add\_\_(baz), but why would you?!
* \_\_init\_\_(): Initializes an object of a class.
* \_\_str\_\_(): Returns a string representation of an object.
* \_\_repr\_\_(): Returns a string representation of an object, used for debugging.
* \_\_len\_\_(): Returns the length of an object.
* \_\_getitem\_\_(): Enables accessing elements of an object using indexing.
* \_\_setitem\_\_(): Enables assigning values to elements of an object using indexing.
* \_\_delitem\_\_(): Enables deleting elements from an object using indexing.
* \_\_iter\_\_(): Returns an iterator object for iteration.
* \_\_next\_\_(): Retrieves the next item from an iterator object.
* \_\_contains\_\_(): Checks if an element is present in an object.

Built-in functions:

* Built-in functions are pre-defined functions provided by the Python language itself. They are available **globally** and can be used without the need for importing any specific modules.
* **range**(1, 4) // 0,1,2,3. Returns a sequence of numbers within a specified range.
* max(): Returns the largest item in an iterable or the largest of two or more arguments.
* min(): Returns the smallest item in an iterable or the smallest of two or more arguments.
* abs(): Returns the absolute value of a number.
* sum(): Returns the sum of all elements in an iterable.
* round(): Rounds a number to a specified precision.
* type(object): Returns the type of an object.
* str(): Converts an object to a string representation.
* list(): Converts an iterable to a list.
* dict(): Creates a new dictionary or converts an iterable of key-value pairs to a dictionary.
* len(): Returns the length of an object (number of items, characters, etc.).
* **input()**: python 3, python 2: raw-input() // Read user input from the command line. input("where are you -->"), get input in cmd mode

name = raw\_input("What is your name? ")

print "Hello, %s." % name

* **exit()** // stops the program
* **print(value, end = ',')** // python 2 does not have the parenthesis
  + **print(r"")** // r ignores escape chars
* **variable.copy**
* **boolean**(value) // returns true if the value is the string

## Format strings

* **format strings : (%s%d%x%i)** // string, digit, hex, integer

print('my name is %s' %name)

print('my name is %s %s' %(name,lastname)) // should be a tuple for multiple varialbes

print('my %s' %name)

* 'Dear {0} {1} {2} '.**format(fn,ln,age)**

## Keywords

* **continue** // ignore the rest of the code, used in loops to end the current execution
* **break** // breaks the loop or function completely
* **pass** // do nothing, is used for when we still don't have any code in function but we define the function.
* **with** // Do something and then close it

with open('natije-2.html','w', encoding='utf-8) as my\_new\_file: my\_new\_file.write(natije.text.strip())

Strings

* string[2] // returns that index value

**myname='Derry'**

**myname[2] // r**

* ''' string ''' // Use three quotes to declare strings
* string**.lower()** // lowercase
* string**.strip()**
* str\*int => repeats str int\_num times
* str.**upper():** Returns a **new string** with all characters converted to uppercase.
* str.**lower():** Returns a **new** **string** with all characters converted to lowercase.
* str.**strip()**: Returns a **new** **string** with leading and trailing whitespace removed.
* str.**split()**: Splits a string into a **list of substrings** based on a specified delimiter.
* str.**join()**: Concatenates elements of an iterable into a single string using the string as a delimiter.
* str.**startswith()**: Checks if a string starts with a specified prefix and returns a **Boolean** value.
* str.**endswith()**: Checks if a string ends with a specified suffix and returns a **Boolean** value.
* str.**replace()**: Returns **a new string** with all occurrences of a substring replaced with another substring.
* str.**find()**: Returns the **index** of the **first occurrence** of a substring within a string.
* str.**isalpha()**: Checks if a string contains **only alphabetic characters** and returns a Boolean value.

Containers

* Objects are either **mutable** or **immutable**. Mutable means we can update it (by parts).
* Strings are immutable

## list

* list\_sample = [value,value1,...]
* list[index] // returns value
* list[1:3] // returns 1,2 and 3 is not included
* list[start index: to index : jump steps] // to-index is not included

**Fruits[0:7:2] // 0, 2, 4, 6**

* value **in** students // returns true, false

**'Derry'.lower() in students // true or false**

* 'Derry' **not in** students

**Methods**

* list.**len**(students)
* list.**append**(' ')
* **lower**()
* list.index(value) // returns index

**list.index('Derry')**

* list.**pop**(index) // default 0, by deafault returns from the beginning. It removes the value from the original list and returns that value.
* list.**remove**(value) // Not index, Doesn't accept indexes, none is returned. It changes the original list.
* list.**reverse**()
* list.**insert**(index,value)

l = [1,2,3,4];

l.insert(2, 5); #[1, 2, 5, 3, 4]

* list.**sort**() // changes the original list

l.sort(reverse=True)

* **filter(function, list)** // with the help of this buil-in function checks if certain values are inside the list. Returns those values in an **iter** that should use list() to be usable as a list.

list (filter(fun,iter)) // to convert it back to list

def is\_even(num):

return num % 2 == 0

list (filter(is\_even, l)) // return [4, 2]

* **map(func, iter)** // applies the given function to each item of a given **iterable** and returns a list of results after.

#list(map(func, iter))

def square(num):

return num \*\* 2

squared\_numbers = list(numbers.map(square))

* **reduce(func, iter)** // In Python, the reduce() function is not a list method like filter() or map(). Instead, it is available in the functools module. The reduce() function is used to apply a specific function to a sequence (e.g., a list) and iteratively reduce it to a single value.

from functools import reduce

numbers = [2, 3, 4, 5]

def multiply(x, y):

return x \* y

product = reduce(multiply, numbers)

print(product)

**Convert string to list and vice versa**

* **list(string)** // returns a list from the given string (or itter or a range object)

list('haha')

['h', 'a', 'h', 'a']

numbers\_range = range(1, 6)

numbers\_range\_list = list(numbers\_range)

print(numbers\_range\_list)

* 'char'.**join**(students) // To cast a string from a list. The items should be a string

','.join(list('haha')) #'h,a,h,a'

* **list.copy()** // create a new one

**L2 = l1.copy()**

**name2 = name // a new pointer, so changes afftect both**

**Set**

* **set(list)** // removes **duplicated** values from the original list and **returns a set** not a list. Need to be converted.

List(set(colors))

## Tuple

* Not mutable
* It has fewer options but it takes fewer resources (memory)
* Names = ('' ,'', '')
* For-in

for i in names

print(i)

* tuple.count()
* tuple.index()
* sorted(tuple)
* sum(tuple) // return the total
* max(tuple)
* min(tuple)
* len(tuple)

## dictionary:

* no indexes
* no key can be duplicated
* {key:value}

students = {'Derry' : 18, 'Rose' : 15}

* student[key] = value // add to dictionary or update if exists. Can't use students.Derry

students['Derry'] = 32

**methods**

* students.**keys**() // returnes a type called **dict\_keys**
* **for** student **in** students = for student in students.keys => Both go through keys

students = {'Derry': 20, 'Rose': 18}

for key in students.keys()

Key + '->' + str(students[key])

for key in students # this also loops through keys

my\_keys = list(students.keys()) # if you want the list type you have to convert it

print(students[my\_keys[st-num]]) # st-num is index

#returns 18

#my\_keys:['Derry', 'Rose']

* students.**get**(a key) // returns value of that key
* students.**popitem**() // takes nothing, removes the **last** **key: value**, and returns it as a **tuple (key, value)**

name, score = students.popitem() # returns a tuple which you can assign to two vairbles

list( students.popitem())

#['Rose', 18]

* students.**pop**(key) => removes the intended key:value, **returns just the value** of that key

Functions:

* should be defined before using
* def fun(parameters): // def: define
* def fun(name='Deafult Value'): //default value is for when we don't pass the value for that parameter which means it is optional. **Optional** arg should be defined **after other args**

def fun(parameter1, parameter2='Deafult Value'):

body

fun(args)

* def fun(**\*args**) // a list of arguments
* def fun(**\*\***reza) // **KWRGS**, can **accept a dictionary** when is called

def multiply(\*args):

body

muplitpy(1,2)

multiply(1,2,3)

def showMyDictionary(\*\*args):

body

showMyDictionary(**name='Derry', lastName='Ram**)

* **anonymous functions** // don't have names just args. They are also called lamda functions.
* **lambda function** // It has one line of code and we should put them in a variable or use them directly where they are expressed, or return them. Very fast compared to normal functions. It is used in **recursive functions and networks**.Lambda functions are typically used in situations where a small, temporary function is needed, and it doesn't require a formal function definition.

X = lambda arg1,arg2,arg3: expressions

def myfun(data):

return lambada var : var\*data # two variables var and data

mul = myfun(10)

mul(2) # returns 2\*10

result = (lambda x: x \*\* 2)(5)

* **pass** // if we don't have any body right now we can use this.

I/O

## File I/O

* Uses **hip** to store files in memory // you can see hip in **process explorer**
* modes // write, read, append
  + read // default
  + write // if not mentioned it is overwritten? What does it mean
* file = open(**'file address'**, modes: **'a'**(append) or **'a+'**(append and create if the file doesn't exist) or **'w'**(overwrite) or **'r'**) // default r

file = open("file.txt", "r") # Open file.txt in read mode

* file**.write(data)**

file = open("file.txt", "w") # Open file.txt in write mode

file.write("Hello, World!\n") # Write data to the file

file.close() # Close the file

* content = file.**read**() // no need to close, close is automated so should be assigned in a variable
* content = file.**readlines**() // returns a list of lines you can loop through [ line1, line2, …]

file = open("file.txt", "r")

content = file.read() # Read the entire file

print(content)

# Or read line by line

for line in file.readlines():

print(line)

* file**.close()** // if we don't close (except file.read()) it means it still is active in that mode that we opened the file
* strip() // to strip the file (trim it)

with open("file.txt", "r") as file:

content = file.read()**.strip()**

print(content)

* replace('\r\n\' , '') // if in the file there are space characters

with open("file.txt", "r") as file:

content = file.read()**.replace**("\n", "")

print(content)

## SQL I/O

* SQL(structured query language) // MySQL, SQL SERVER, MongoDB, Hadoop, CouchDB (nodejs)

**SQLite**

* A small database // for example in android creates a database on the client side
* It's a built-in library
* Import sqlite3
* conn = sqlite3.**connect**(address) //

conn = sqlite3.connect('pythonDB.sqlite') # it's better to include the extention so other systems can understand it.

* **cursor** // our connection brigde to the SQLite

c = conn**.cursor** # c is the cursor

c.execute('CREATE TABLE students\_table ()')

* curser.**fetchall**
* cursor.**fetchmany**(num) // default 1, returns one, every time called, and the next time that is called returns the next one.

for i in cursor.fetchall(): // i is row

for j in i: // a list of tuples, every row has many columns

print j // j is column

* cursor.**arraysize**

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect("mydatabase.db")

cursor = conn.cursor()

# Set the arraysize to fetch 5 rows at a time

cursor.arraysize = 5

# Execute a SELECT query

cursor.execute("SELECT \* FROM mytable")

# Fetch and process rows based on the arraysize

while True:

rows = cursor.fetchmany()

if not rows:

break

for row in rows:

print(row)

# Close the cursor and connection

cursor.close()

conn.close()

* Sql injection attack // There is no prepare method for sqlite in python. Using parameterized queries with placeholders helps prevent SQL injection attacks and improves the readability and reusability of your code by separating the SQL logic from the parameter values.
  + ? and execute method

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect("mydatabase.db")

cursor = conn.cursor()

# Prepare the SQL statement with placeholders

sql = "SELECT \* FROM mytable WHERE column1 = ? AND column2 = ?"

# Execute the prepared statement with values

cursor**.execute**(sql, ("value1", "value2"))

# Fetch and process the results

rows = cursor.fetchall()

for row in rows:

print(row)

# Close the cursor and connection

cursor.close()

conn.close()

**running SQL code**

* **conn.commit()** // usually we don't need it for select

import sqlite3

# Connect to the SQLite database

conn = sqlite3.connect("mydatabase.db")

cursor = conn.cursor()

# Execute SQL statements that modify the data

cursor.execute("INSERT INTO mytable (column1, column2) VALUES (?, ?)", ("value1", "value2"))

cursor.execute("UPDATE mytable SET column1 = ? WHERE column2 = ?", ("newvalue", "value2"))

# Commit the changes

**conn.commit()**

# Close the cursor and connection

cursor.close()

conn.close()

**close conn**

* **conn.close()** // for every conn.connect()

**Query tips:**

* SQL queries are not case-sensitive but the **name of tables all should be in lowercase**
* int // It's better to don't set the length since it is not a number but the byte that it uses.
* id // Always UNSIGNED

import sqlite3

# Connect to or create a SQLite database

conn = sqlite3.connect("mydatabase.db")

# Create a cursor object to interact with the database

cursor = conn.cursor()

# Create a table

cursor.execute("CREATE TABLE IF NOT EXISTS mytable (id INTEGER UNSIGNED PRIMARY KEY AUTOINCREMENT, name TEXT, age INTEGER)")

# Commit the changes (if any)

conn.commit()

# Close the cursor and connection

cursor.close()

conn.close()

* INSERT INTO tablename ( column1 , column2) values("string", "")
* int: type, increment: attribute
* UPDATE users-tbl SET password = "123" where id = 1
* **DELETE** from users-tbl WHERE id = 2
* **DROP TABLE** users-tbl'
* **TRUNCATE**
* **os.remove** // to remove the db file

## MySQL

* install mysql-connector-python
* import mysql.connector
* Enum // for example male/female
* **MYISAM** engine // how to store the values, for example with comma
* ping localhost -6 or -4 (ip version) // ip is better, not host name

**Connect and create:**

* TABLES = {} // create a dictionary to put all tables inside it.
* TABLES['mft-python']=( ) // parentheses are used when the string is long and in multiple lines
* my-connection = MySQL**.connector.connect**(user='', password:'', host:'', database:'') // to connect with DB user, pass, host, database => connection string
* my-cursor = my-connection.**cursor**()
* my-cursor.**execute()**

package

* why?
  + Organize code
  + Code management
  + Clean coding
* \_\_init\_\_.py // a file by this name. To create a package inside this file we import modules.
  + This file is not needed in 3.3 or above
* Don't need to export anything. You can import vars and function and classes by import keyword.
* import file\_address

import sayHello

sayHello.hi("Derry")

* from file\_address import \*(all functions and variables)

from sayHello import \*

hi("Derry")

from os import \*

* from .file import \* // python 3 needs the dot when creating a package

mypackage/

\_\_init\_\_.py

module1.py

module2.py

subpackage/

\_\_init\_\_.py

module3.py

* from .file import \* // When you import a package or module, Python looks for an \_\_init\_\_.py file in the package directory. If it exists, Python executes the code within the \_\_init\_\_.py file. This allows you to perform any necessary setup, initialization, or define package-level elements before the package or module is used.
* An **\_\_init\_\_** example:

# Import specific modules or variables to make them accessible directly from the package

from .module1 import some\_function

from .subpackage.module3 import another\_function

# Define the \_\_all\_\_ list to control what gets imported when using 'from package import \*'

\_\_all\_\_ = ['some\_function', 'another\_function']

# Define package-level constants

PI = 3.14159

DEFAULT\_TIMEOUT = 10

# Initialization code

print("Initializing mypackage...")

# Other package-specific code or configurations

* Inside main: Here we don't need a dot before package1 or will throw an error.

from package1 import module1

print(module1.multiply(4,1))

print(module1.random\_var)

* Inside package1/module1.py:

def multiply(x,y):

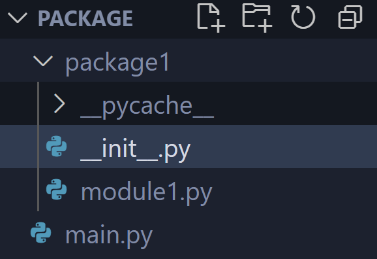
return x\*y

random\_var = "Hello World"

* Inside \_\_init\_\_.py of package1/ here dot is needed

from .module1 import multiply

print(multiply(3,9))



Error handling

* **try/catch** // the parts that we suspect may encounter some errors we put in a try-catch
* **try**
* **except** for exception as err. Exception handling.

except Exception as err:

print(err)

* else
* finally
* To create cleaner code it's better to use it for when functions are called

try:

# Code that might raise an exception

result = 10 / 5

except ZeroDivisionError:

# Code to handle the specific exception

print("Error: Division by zero is not allowed.")

else:

# Code that is executed if no exception occurs

print("The result is:", result)

finally:

# Code that is always executed, regardless of an exception

print("End of the try-except block")

RegEx

## re.compile

* **import re** // import the module
* pattern=re.compile(reza, re.IGNORCASE) // second prameters is flags in compile
* pattern=re.compile('\w+')
* re.match(pattern, text) // returns a match object

import re

pattern = r'^\/Sousan(?:\/\w+)\*$'

text = '/Sousan/Derry/Rose/Danny'

matches = re.match(pattern, text)

#output: <re.Match object; span=(0, 24), match='/Sousan/Derry/Rose/Danny'>

print(matches)

if matches:

captured\_names = tuple(matches.group(0).split('/'))

print(list(filter(None, captured\_names))) # Output: ['Sousan', 'Derry', 'Rose', 'Danny']

* pattern.findall(text) // returns an array

print(re.findall(r'(\w+)', text)) # Output: ['Sousan', 'Derry', 'Rose', 'Danny']

OOP

* **Spaghetti code** // the style of coding we used to write, was very bad
* **Function base** // This was better but still had issues
* **MVC** // programming styles. It is a **software design pattern**
* **HMVC** // stands for **Hierarchical** Model-View-Controller, a software architecture pattern that extends the MVC (Model-View-Controller) pattern by adding **multiple layers of controllers and views**.

## Class

* Class is the **model** and object is the instance that we make out of that model.
* Variable set outside \_\_init\_\_ belongs to the class. They're shared by all instances.
* Variables created inside \_\_init\_\_ (and all other method functions) and prefaced with self (self.name) belong to the object instance.

class Car:

pass

benz = Car()

* **Methods** and **properties/attributes** of a class
* **\_\_init\_\_(self)** // **initialize**. When making an object a method is called automatically. It is a reseved method in python classes. It is called as a **constructor** in object oriented terminology. This method is called when an object is created from a class and it allows the class to initialize the attributes of the class.
* To differentiate a method like init from other methods, we use --init-- in the method declaration.
* No need for new keyword to create an instance.

class Car:

def \_\_ini\_\_(self):

Print('car is made')

car\_1 = Car() # init is called

* **self** // When we declare a method for a class, we pass a variable called self which is a reserved keyword and the value is the object name. All methods should have it. We can use any name for it instead of self. Most object-oriented languages pass this as a hidden parameter to the methods defined on an object; Python does not. You have to declare it explicitly.

class Car:

def \_\_ini\_\_(poo):

self.color = 'blue'

self.speed ="300"

self.brand='benz'

self.fuel = 'gas'

def run\_forward(poo):

print('self is going forward')

def stop(poo):

print('self is going forward')

def info(poo):

print(self.rang)

def status(self):

print('status')

## Basic concepts in oop

* **Abstraction**
* **Polymorphism**
* **Encapsulation**
* **Inheritance**

**Inheritance**

* class child(parent)

class Human:

def \_\_init\_\_(self):

self.sex = ''

class Mother(human):

pass

class Child(mother)

pass

**Abstract classes**

* **abstract classes** // An abstract class can be considered **a blueprint for other classes**. It allows you to **create a set of methods that must be created within any child classes** built from the abstract class. **A class that contains one or more abstract methods is called an abstract class.** An abstract method is a method that **has a declaration but does not have an implementation**. While we are **designing large functional units** we use an abstract class. When we want to provide **a common interface for different implementations of a component**, we use an abstract class.
* **from abc import ABC, abstractmethod** // first the class should inherit from ABC or we can't make an instance out of it directly. Because some classes should at least have a method. So we force it.
* **@abstractmethod** // with abstract we force the children to have that method by overriding the abstract methods of the parent. **If they don't, you can't have a direct instance of them.**

from abc import ABC, abstractmethod

Class Human(ABC):

**@abstractmethod**

def run(self)

pass

Class daughter(Human):

def run(self)

print('child run')

**polymorphism**

* How to code to work with **any kind of data type**.
* The method added in the example below should have **different functionality for different data types.** So for every class, you have to override the base method and add

a =1, b=2

a='derry', b='rose'

a--add--b

**Encapsulation**

* getter
* setter

import random

class library:

def \_\_init\_\_(self):

pass

def showBooks(self):

pass

def books(self):

books\_num= random.randint(1,10)

class customer:

def \_\_init\_\_ (self):

pass

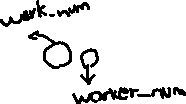
Multithreading

* import threading
* import queue // queue is like list, but it has put() and get() and gete() will delete the item

**for the example below:**

* make threads as many as the workers
* **threading.lock** // when we have a shared resource like print, it locks that resource and doesn't allow any other worker to use it. Instead of using it directly, we create a lock and every worker can use it when it's needed.
  + **lock.acquire**
  + **lock.release**
* **boss** // give workers work till we still have work to do
  + my\_forghoon**.empty()** => true, false
* **task\_done()** // locks and sends a notification until the work is done. because the queue has this problem that when it's empty it still locks. Even though this problem is because to have higher speed. **forghoon.join()**, it doesn’t run the rest of the program while we still have a task inside it. If we don't have task\_done, empty and join won't run. task\_done() Indicate that a formerly enqueued task is complete. Used by queue consumer threads. For each get() used to fetch a task, a subsequent call to task\_done() tells the queue that the processing on the task is complete.
  + **forghoon.get()** // if it's empty program will stuck here
* **setDaemon** // If the program faced any problem, it doesn't become a zombie. If the program crashed all threads die too.
  + **self.\_timer.daemon** = True // python3.10 abandon this usage "setDaemon(True) "
* It's better we first give the worker (threads) and then send the job (queue). But in this scenario, you should make the while true by default.
* How many workers should we have? a number that avoids idle times and considers the CPU

50 /10 = 5 // parallel jobs



import time

import threading

from queue import Queue

start\_time = time.time()

forghoon = Queue() #instead of creating a list to put the tasks in it

my\_lock = threading.Lock() #create a lock

#the job itself, we use the print resource to show a value

def job(a):

time.sleep(1)

brick = a # a is 0 to 49

# use the lock and lock the print which is our resource

with my\_lock: # here we print [0 \*\* 2 ] = 36 up to [49 \*\* 2 ] = 1849

print("[" + str(brick) + ' \*\* 2 ] = ' + str(brick \*\* 2) )

def boss():# the boss will give the tasks to workers

# we can also use while not forghoon.empty():

while True: # since new tasks are added to queue at any time. Keeps assigning jobs forever until we don't have works to do anymore. This loops exists because of deamon = True

job(forghoon.get()) #get a job from queue and do it

forghoon.task\_done() # the task is done after job function is done

#first create 10 workers. This will run. It's not a function

#target is the same for all of them. The boss will give the jobs as new jobs are added to queue(next for loop). The while True makes sure of that.

for num in range(10):

my\_worker = threading.Thread(target=boss) #create a worker

my\_worker.daemon = True # Don't become a zombie

my\_worker.start()

#Here we are adding jobs to queue to do as time goes by

for i in range(50): # put the jobs in a queue

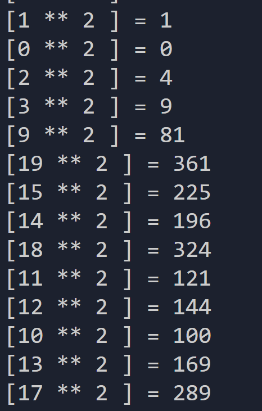
#100 121 144 .....

forghoon.put(i)

forghoon.join() # don't run the rest until we have task

end\_time = time.time()

print( "Total Time:" + str(end\_time - start\_time))

* The threads created using threading.Thread are marked as **daemon** threads with my\_kargar.setDaemon(True). Daemon threads are background threads that do not prevent the program from exiting if all non-daemon threads have completed. Since your main thread doesn't have any further work to do after creating and starting the worker threads, it will exit naturally once all the worker threads have completed their tasks. The remaining worker threads will be terminated because they are marked as daemon threads. So, even without an explicit condition in the while loop, the program will exit after all the tasks in the forghoon queue are processed because the worker threads are daemon threads.
* The forghoon**.join()** function call in the given code is used to **synchronize** the main thread with the worker threads. In the code, forghoon is an instance of the Queue class, which is a thread-safe data structure. The join() method of the Queue class is used to **block the main thread** (the code in the main script file or the code within the if \_\_name\_\_ == "\_\_main\_\_": block) **until all tasks in the queue are completed**. When forghoon.join() is called, the main thread will wait and not proceed further until all tasks in the forghoon queue have been processed and marked as done using forghoon.task\_done(). The task\_done() method is typically called by the worker threads to indicate the completion of a task. By using forghoon.join(), **you ensure that the main thread does not exit prematurely before all tasks have been processed by the worker threads.** It provides a synchronization point where the main thread waits for the worker threads to finish their work. Using forghoon.join() is important to ensure that all tasks are completed before the program continues or exits, especially if you have additional work or dependent operations that should occur after the worker threads have finished their execution.
* They are not done in order

## Multi-process

* When the **threading is no longer allowed** by the operating system we use GPU to create processes. Gup -> cud? Not sure what this is

Logging-system hacking

* **Bruteforce** // consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found.
* We get one of the HTTP requests sent in the network.
* Raw headers // copy/paste edit it and we see what we need in that request and send our made request to the URL, the data should be a dictionary

import requests as req

import threading // these two modules can be used to send the requests

data = {

'email' : 'a….com',

'password' : '100',

'registerlogin': 'login',

'axz..': ''

}

import requests

# The API endpoint

url = "https://jsonplaceholder.typicode.com/posts/1"

# A GET request to the API

response = requests.get(url)

print( response.headers)

import requests

BASE\_URL = "https://example.com/api/content/v1/products/search"

headers = {

"Connection": "keep-alive",

"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"}

params = {

"page": 20,

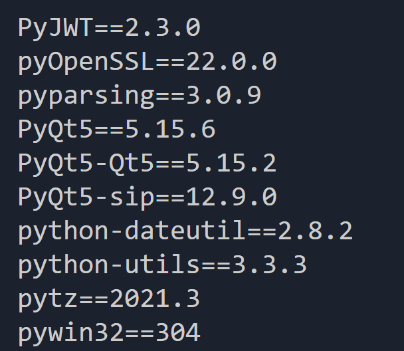
"page\_size": 25,

"type": "image"}

response = requests.get(BASE\_URL, headers=headers, params=params)

Project virtual environment

* Python **pyinstaller** //.PyInstaller reads a Python script written by you. It analyzes your code to discover every other module and library your script needs in order to execute. Then it collects copies of all those files – including the active Python interpreter! PyInstaller lets you freeze your python application into a stand-alone executable. This installer supports Linux, macOS, Windows, and more; and is also compatible with 3rd-party Python modules, such as PySide6. PyInstaller and virtual environments are two separate tools that can be used together for managing and distributing Python applications.
* A virtual environment, on the other hand, is a tool that helps create isolated Python environments with their own dependencies. It allows you to install and manage specific versions of packages for each project, without interfering with other projects or the system-wide Python installation.
* When using PyInstaller within a virtual environment, you can ensure that the dependencies required by your Python script are properly captured and bundled into the executable. By activating the virtual environment before running PyInstaller, you can ensure that it uses the packages installed in that environment, rather than the system-wide packages. This helps create a self-contained executable that includes all the necessary dependencies from the virtual environment.
* pip freeze // The command pip freeze is used in Python to generate a list of installed packages and their versions. It provides a snapshot of the current state of installed packages in your Python environment.



* **pip install -r requirements.txt** // The command pip install -r requirements.txt is used in Python to install packages listed in a requirements.txt file. It allows you to easily recreate a specific Python environment with all the required packages and their respective versions. in requirements.txt:

numpy==1.19.5

pandas==1.3.0

matplotlib==3.4.3

* **python3 -m venv** // to create a project we make a virtual environment. For this, we only need the required modules and use virtual environment to manage and run those modules. The command python3 -m venv is used to create a virtual environment in Python 3. A virtual environment is an isolated Python environment that allows you to install packages and manage dependencies specific to a project without interfering with the global Python installation or other projects. It provides a clean and separate environment where you can install packages and control the Python version used. When you run python3 -m venv <directory\_name>, it creates a new virtual environment in the specified <directory\_name>. This directory will contain all the necessary files and directories for the virtual environment. For example, running python3 -m venv myenv will create a virtual environment named "myenv" in the current directory. Inside the "myenv" directory, you will find subdirectories such as "bin" (or "Scripts" on Windows), "lib", and "include", along with other configuration files.
  + To activate the virtual environment, you need to run the appropriate activation script depending on your operating system. For Unix/Linux systems, the command is usually source myenv/bin/activate, and for Windows, it is myenv\Scripts\activate.bat. Activating the virtual environment modifies the system's PATH variable, ensuring that the Python interpreter and installed packages from the virtual environment are used.
  + Once the virtual environment is activated, you can use pip to install packages, run Python scripts, and manage the environment's dependencies. It provides a clean and isolated environment where you can work on your project without conflicts with other Python installations or projects.

python -m venv tutorial-env

tutorial-env\Scripts\activate.bat

deactivate // run this

pyinstaller: Convert py files to exe

* Python installer with fbs // The combination of Python installer and fbs refers to a method of packaging and distributing Python applications using the fbs framework.
  + Python installer tools, such as PyInstaller, cx\_Freeze, or PyOxidizer, are used to convert Python scripts into standalone executables that can be run on different platforms without requiring a Python interpreter or additional dependencies. These tools bundle the Python interpreter and all necessary dependencies into a single executable file.
  + fbs (Frozen Binary Software) is a Python framework that simplifies the process of packaging and deploying Python applications. It provides a set of tools and conventions to create cross-platform applications with a graphical user interface (GUI). fbs uses PyInstaller as its default bundling tool, but it abstracts away the complexities of configuring PyInstaller and provides a higher-level interface to simplify the packaging process.

## How

* Set up a virtual environment (optional but recommended):
  + Create and activate a virtual environment using python3 -m venv <venv\_name> and source <venv\_name>/bin/activate (Unix/Linux) or <venv\_name>\Scripts\activate.bat (Windows).
* Install the required dependencies:
  + Install the fbs framework by running pip install fbs.
  + Install any additional dependencies your application requires.
* Set up the fbs project:
  + Create a new directory for your project and navigate to it.
  + Run fbs startproject to initialize an fbs project structure.
  + Modify the project structure and configuration files according to your application's needs.
* Develop your application:
  + Write your Python application code, including the graphical user interface (GUI) using a suitable GUI framework (e.g., PyQt or PySide).
* Package and distribute your application:
  + Run fbs freeze to bundle your application using PyInstaller.
  + Use fbs installer to create an installer or distributable package for your target platform.
  + Follow the prompts and provide necessary information during the packaging process.

## Create exe with pyinstaller

* In cmd
  + pip install pyinstaller
  + pyinstaller 02.py // gives a set of files
  + pyinstaller --onefile 02.py // only one file
  + pyinstaller -w --onefile 02.py // In the context of the PyInstaller command pyinstaller -w --onefile 02.py, the -w option stands for "windowed" mode. When you use the -w option, PyInstaller generates an executable that runs without displaying a console window or terminal. This is particularly useful for creating graphical user interface (GUI) applications where you don't need or want a visible command-line interface
* For qtcore.dll error when using pyinstaller

pyinstaller --hidden-import=PyQt5.QtCore,PyQt5 Library.py

pyinstaller --add-data C:\Users\Asus\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\PyQt5\Qt\bin;Qt5Core.dll --paths C:\Users\Asus\AppData\Local\Programs\Python\Python37-32\Lib\site-packages\PyQt5\Qt\bin --hidden-import=PyQt5.QtCore,PyQt5 Library.py

Modules

## os module

* Is used to create a virus, handling problems in operating systems
* **os.getcwd** // current process. absolute path of the current working directory.
* **os.mkdir**("seasons")
* **os.rmdir**("file") // Delete an empty dir
* **os.listdir**('.') // returns a list of files and folders . dot is the current dir

for i in os.listdir("."):

if("season" in i): # if there is the file or folder named season

os.rmdir(i) # i is dir

* **os.system**(dir) // If argument is a string, it passes it to the shell to perform it.
  + **os.system**('explorer .') // opens the explorer where we are

os.system(dir)

os.system("ls")

os.system("remove c:\\")

* **os.name** // The os.name attribute in Python provides information about the name of the operating system. It returns a string that represents the name of the current operating system.
* **os.walk**('.') // lists all dir and subdirs. Returns a **generator** type

import os

for my\_folder, my\_folder\_folders, my\_folder\_files in os.walk('.'):

print(my\_folder)

if(len(my\_folder\_folders)):

print(" "+str(', '.join(my\_folder\_folders)))

* **os.path.abspath**(folder) // returns absolute path(complete path) c://../folder/file.txt

## request module

* <https://docs.python-requests.org/en/master/>
* one of the best modules to work on web
* **pip install request**
* **import request as req**
* my-result = request.**get**(my-site)

target\_url= "url"

my-result = request.get(my-site)

my-result.text

# result.request.headers

# my\_headers = { 'user\_agent': 'firefox'} // so it see us as a browser

result.request.headers.get(address , headers={}, verify=false)

a = open('result.html','w', encoding='utf8')

a.write(my-result.text)

a.close

**Kavenegar**

* api.kavenegar.ir/sendsms/phone-number/message
* req.get('api.kavenegar.ir/sendsms/091951501/salam')

**Api**

* Restful(json), Soap(XML) // restful is like dictionary with key:value

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>

...

</soap:Header>

<soap:Body>

...

<soap:Fault>

...

</soap:Fault>

</soap:Body>

</soap:Envelope>

* API key // To work with APIs we need passwords to connect to the API. Look into the api's documentation.
* APIs:api.openweathermap.org, APIs made in Iran,
* Weather API // how to **parse XML** (key\_value and sub\_key\_value in tag syle) with python. We Can do it with **beautiful soup**

import requests as req

import colorama

from bs4 import BeautifulSoup as bs

colorama.init

base\_url ="http://parsigoo.ir/api?serviceType=weather-API&q=" # returns xml

city\_name = input("where are you -->")

result = req.get(base\_url + city\_name, verify=False) # use verify false because it may block you

my\_soup = bs(result.content, "xml")

status = my\_soup.find\_all('status')

try:

print(status[0].get\_text())

with open('result.txt' ,'a', encoding='utf-8') as my\_file:#don't forget the encoding

my\_file.write(str(status) + "\n\r")

except:

pass

**Json**

* **res.json()**
* result.keys()
* result.values()
* a json of jsons

{'names':('ali','reza','hossein')}

* in python, we use json for the **configure files**. config.json:

{

'user': 'admin—',

'pass': '…',

'myName': ''

}

* **json.load()**

with open('config.json','r') as myfile:

my\_config = json.load(myfile)

}

for word in my\_result.json():

print(word['text'])

my-file.write(word['text'] + '\r\n') # write in a file

## colorama module

* to have colorful code output
* import colorama
* colorama.init()
* print(colorama.Fore.LIGHTCYAN\_EX())

import colorama

from colorama import Fore, Back, Style

colorama.init()

print(Fore.LIGHTCYAN\_EX + 'This text is light cyan.')

print(Fore.RED + 'This text is red.')

## BeatifulSoup

* **Data analysis**
* **Web Scraping** with Python Using Beautiful Soup
* beautifulsoup tutorial // website
* soup = **BeautifulSoup**(webResult, "html.parser")
* soup.**findAll**(tag)
* soup.**find**(tag) // the first one found
* soup.**find**(tag).get(attribute) // the first one found

import requests

from bs4 import BeautifulSoup

my\_site='https://pyinstaller.org/en/stable/operating-mode.html'

my\_result = requests.get(my\_site)

if(my\_result.status\_code == 200):

print(" successful ")

soup = BeautifulSoup(my\_result.content , "html.parser") # content is big don't print it. The second param can be 'xml'

# soup.findAll('a') # find all links

my\_result.text.title()

# print(soup.findAll('a'))

print(soup.findAll('a')[2].get('href')) # to get all use loop

* status\_code, response codes:
  + **404** // notfound, the page doesn't exist
  + **200** // ok, successful
  + **301,302** // moved permanently, moved temp
  + **500** // internal server error
  + **400** // bad request
* To add Persian to a file use utf-8 always.

## Time module

* import time // built-in module
* time.**time**() // now in seconds
* time.**strttime**("%Y") // %m %d

import time

current\_year = time.strftime("%Y")

print(current\_year)

* time.**sleep**(0.2)
* import jdatetime // shamsi
* jdatetime.datetime.now()

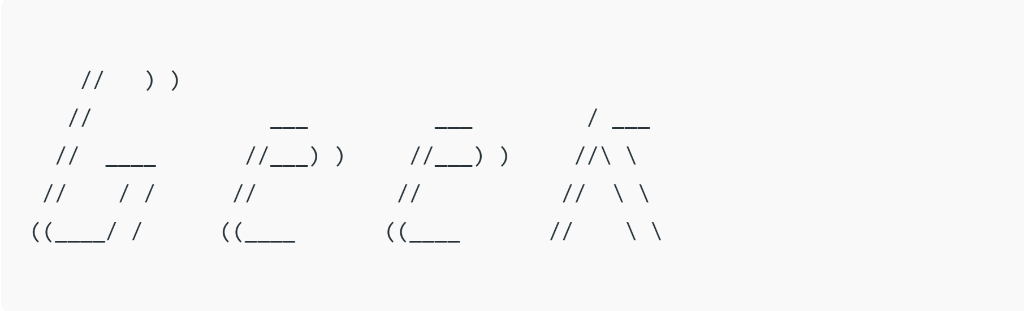
## pyfiglet module

* pyfiglet takes ASCII text and renders it in ASCII art fonts.
* **pip install pyfiglet**

import pyfiglet

result = pyfiglet.figlet\_format("G e e k", font = "alphabet" )

print(result)



Data science modules

## Sites and youtube:

* Youtube --> Jabrils // AI and Machine learning, Linear algebra
* [Introduction - Learn Python for Data Science](https://www.youtube.com/watch?v=T5pRlIbr6gg)
* <https://www.csdojo.io/data> // some sample data for data science, csv sheets
* https://www.kaggle.com/

## Modules

* matplotlibrary // draw charts
* pandas // works with the matplotlibrary, and is used to work with data. It can read pdf, word, XML, html. Run it in Jupiter to see the tables in a completely visual way.

import pandas as pd

my\_data = pd.read\_csv('sample\_data.csv')

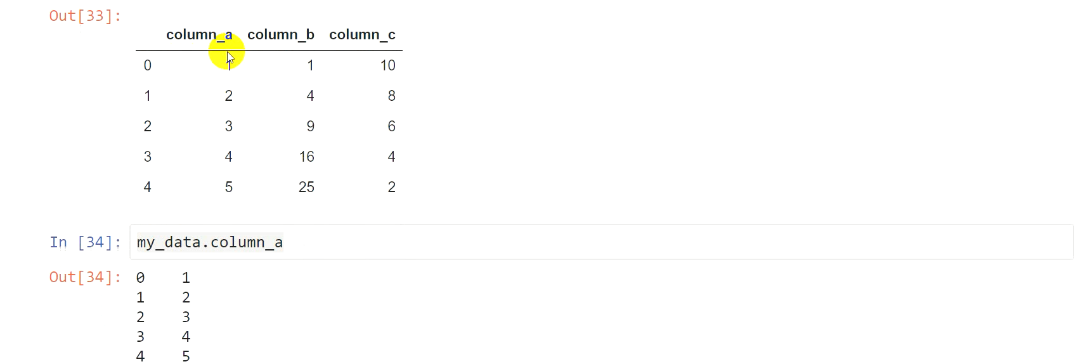
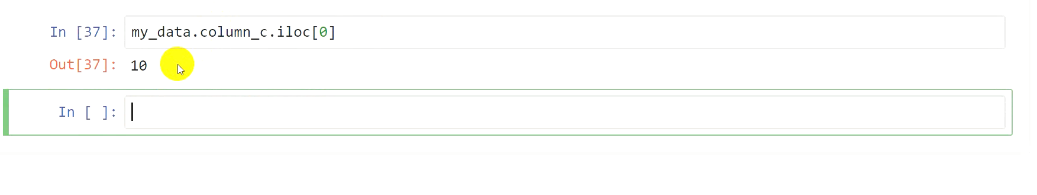
#pd.read\_json()

#pd.read\_html()

#Pd.read\_excel()

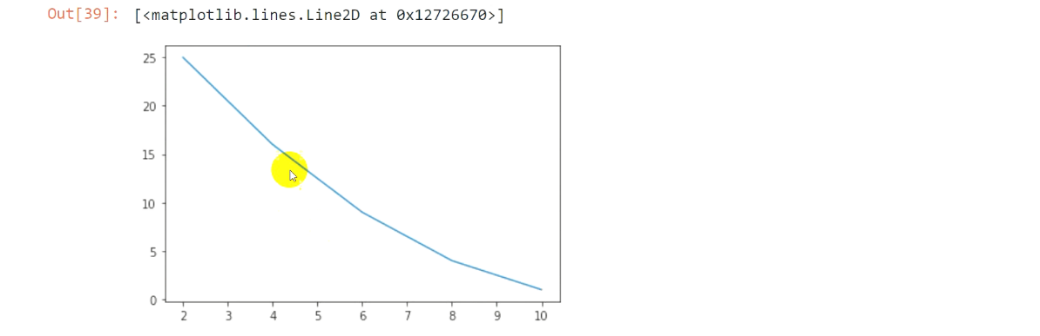
Print(my\_data.column\_a) # column\_a is the column name.

Print(my\_data.colum\_c.iloc[0]) # the value of the that colum in row 0



import matplotlib import pyplot as plt

plt.plot(my\_data.colum\_b , my\_data.colum\_c) # (x,y) on a two-dimensional plane



## Compare data

* Countries from csdojo
* conda install pandas // Installing pandas with Miniconda
* pip install pandas // pandas can understand different extensions

functions:

* **read\_csv(file)** //
* **plt.plot(x,y)**

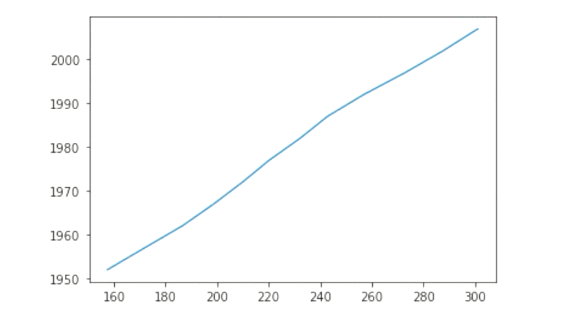
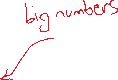
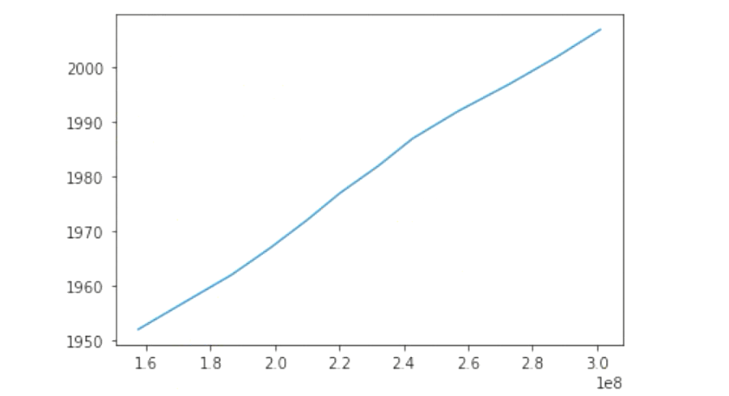
my\_data = pandas.read\_csv(countries.csv)

data\_us = my\_data[ my\_data.country == 'United States']

data\_china = my\_data[ my\_data.country == 'China]

plt.plot(data\_us.population/10 \*\* 6, data\_us.year) # To handle the big numbers we devide it to a power of 10, 10\*\*6 gives million

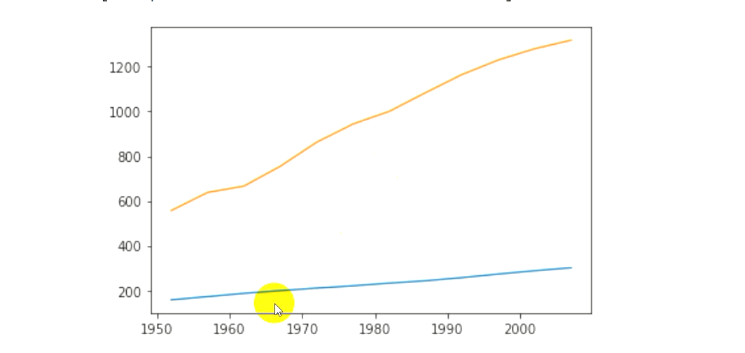
* Y axis is the year and x is the growth data

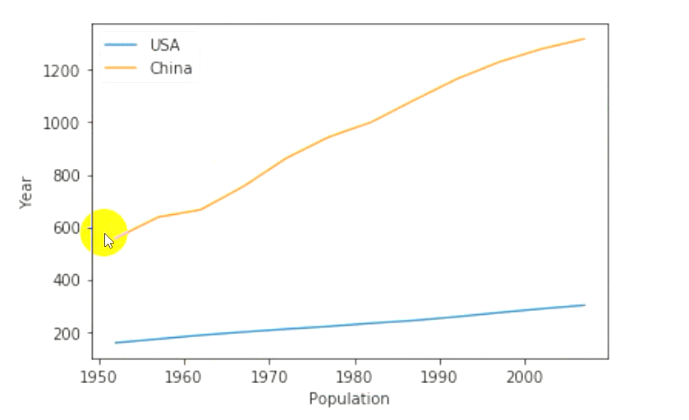


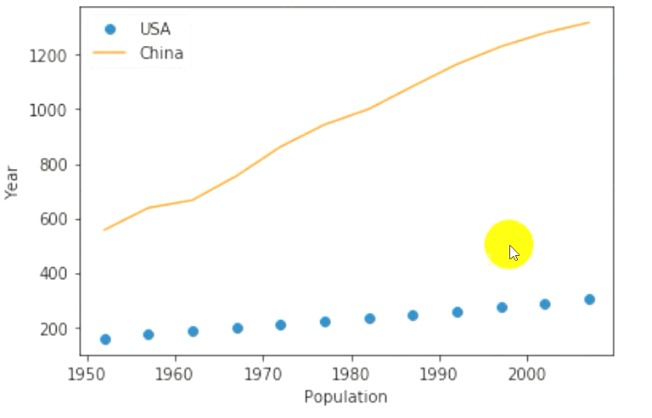
* Compare two countries

plt.plot(data\_us.year,data\_us.population/ 10 \*\* 6)

plt.plot(data\_china.year,data\_us.population/ 10 \*\* 6)



* plt.**xlabel**("Population")
* plt.**ylabel**("Year")
* plt.**legend**(["USA", "China"])
* plt.**plot**(…, …, **modes**) // modes: piechart, dots.
  + 'o': separate dots



import pandas as pd

import os // for address

my\_data = pd.read\_csv('file') # read\_csv, read\_xml

my\_data.colum\_a.iloc(0) # my\_data.columnName.row

my\_data[my\_data.country == 'united sates' ]

from matplotlib import pyplot as pl

x =[10,20,30]

y= [5,2,17]

pl.plot(x)

pl.plot(y)

pl.plot(x,y)

pl.ylabel()

pl.xlabel()

plt.legend(["a"],["b"]) # for what category china and usa for example

Image processing moduels

* pip install numpy // to work with pictures and two-dimensional arrays. Since the pixel is two dimensional (x,y), we can use this to create images.
* pip install opencv-python // [opencv-tutorials.html](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_tutorials.html)
  + imread(picAddress, modes) // import pitcure. mode for example cv2.IMREAD\_GRAYSCALE, cv2.IMREAD\_COLOR. grayscale is processed faster.
  + imshow(' ', file)
  + cv2.waitkey(0) // if we don't use this the picture closes immediately, but using this you need to press a key.
  + cv2.destroyAllwindows() // Frees up hip memory
  + cv2.videoCapture(0) // camera. 0 is the camera number. We only have one camera.
  + cv2.**cvtColor**(frame, **cv2.COLOR\_BGR2GRAY**) // Opencv 🡪 RGB-BGR // the engines of openCv is not RGB but BGR. To convert BGR2GRAY.

import cv2 # picture and webcam

import numpy # two dimensional arrays

from matplotlib import pyplot

# # picture

# my\_picture = cv2.imread('image.jpg', cv2.IMREAD\_GRAYSCALE)

# cv2.imshow('my window', my\_picture) # my window is the name of the window we want to open.

# camera

my\_webcam = cv2.VideoCapture(0) # puts the webcam in my\_webcam

while True: # to referesh the camera

ret, frame = my\_webcam.read() #/ it gives a tuple, frame is the framerate

my\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)# my\_frame is the changed frame. You can change the frames. For example the color.

cv2.imshow('test', my\_frame)

if cv2.waitKey(1) & 0xFF == ord('q'): # to press q key to close break. & is the bit wise with number 255

break

cv2.waitkey(0)

cv2.destroyAllwindows() # use this two method together

* How to save the camera video

my\_webcam = cv2.video.capture(0)

fourcc = cv2.VideoWriter-fourcc(\*'XVID') #codec. The best codec is avi

out = cv2.videowriter('output.out', fourcc, 20.0, (640,4080)) # what file to store in, 20 framerate and the last value is the size of video

my\_webcam = cv2.videoCapture(0)

while True:

ret, frame = my\_webcam.read()framerate

my\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY

out.write(frame) # write

cv2.imshow('test', my\_frame)

if cv2.waitkey(1) & 0xFF == ord('q'):

cv2.waitkey(0)

cv2.destroyAllwindows()

## Draw shapes

* cv2.**line**(picture, tuple , tuple , tuple, number) // start point [x,y], second point, color, thinkness

cv2.line(frame, (100,200), (400,500), (255,0,0),3)

* cv2.**rectangle**(onwhat, topleftpoint, bottomleftpoint, color, thickness)

cv2.rectangle(frame,(200,300),(400,500),(0,0,255), 4)

* OpenCV face detection //
  + [face-detection-with-opencv-and-deep-learning/](https://www.pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/)
  + In GitHub there are some XML files, that have all the face, lips, and eyes standard detection, and you have to import it and use OpenCV detection.
* Text on img

my\_font = cv2.Font\_HERSHEY\_COMPLEX

cv2.putText(frame, 'hi', (200, 300), my\_font, 5, (255,0,0)) # 5 is the size and the last one the color

QT-5 module // desktop visual app

* Pyqt // Works with **events**
* [15 Minute Apps — Common desktop apps in Python, using PyQt](https://www.pythonguis.com/examples/15-minute-apps/)
* Qt designer // a cross-platform program to graphically design apps visuals
* pip install pyqt5
* pip install pyqt5-tools
* pyvic5 -x ol.ui -o ol.py // convert ui to py

Main windos

Centeral widget

toturialspoint