

1 - Some string notes :-

\b = Backspace

\n = new line

\” = put “” to the string

\t = tab space

\xhh = hexadecimal value of char

\r = will replace word after “r” with words before it and in file handling give the true path for file

EXAMPLE : `print(“123456\rhello”) = hello6`

Strings :-

2 - String is array of characters **“REMEMBER IT!”**

3 - **Mystring[0]** = first char of word

Mystring[-1] = last char of the word “first from end”

Mystring[Strat:End:steps]

4 - **Len(mystring)** = the length of the string starting from 1 to n

5 - **Strip() = rstrip() = lstrip()** = removing spaces from the string (r tends to right and l tends to left)

Parameters of strip(“put the symbols or chars you want to remove)

6 - **Title()** = convert all start words Capitalized and also every char after number

7 - **Capitalize()** = convert all words after numbers to small if it capital and vice versa

8 - **Zfill()** = fill number with zeros in start based on the length of parameter

9 - **Upper()** = make the word uppercase

10 - **Lower()** = make it lowercase

11 - **Split()** = split it into list

You can control your like split(separator = “”, max words in to get split = “”)

12 - **Rsplit()** = do the same of 11 but from the right

13 - **Center()** = trying to make the word in the center “Center(length=number, Sep)”

14 - **Count()** = by putting your word in Count will find it and will add 1 to it’s counter when it get found

15 - **Swapcase()** = make every upper to lower and vice versa

16 - **Startswith()** = Boolean value if you want to check the word if it starts with it or not

17 - **Endswith()** = Boolean value if you want to check the word if it with it or not

18 - **Index()** = return the index of the char , and if it doesn’t in it return error

19 - **Find()** = do the same to index , but the difference if the char not in the word return -1

20 - **Rjust() and Ljust()** = put the symbols to the name you want ,EX : `string.Rjust(10, “”)`

he puts to the length of the word with symbols* and output = string***

21 - **Splitlines()** = return all lines which get split into list

22 - **Expandtabs()** = put length of tabs for word

23 - **Istitle()** = return if the string is title

- 24 - **isspace()** = return if the string has spaces
- 25 - **isupper()** and **islower()** = return if the string upper or string is lower()
- 26 - **isidentifier()** = return if the word can be a variable
- 27 - **isalpha()** = return if the string just from a to z string without anything else
- 28 - **isalnum()** = return if the string just from a to z string and numbers
- 29 - **Replace()** = change old value into new value and you can put number of changes
- Join(Iterable)** = return all the words in list and concatenate it with symbol

Ex : `list = ["Mahmoud", "Essam", "Fathi"]`
`"-".join(list) = Mahmoud-Essam-Fathi`

- 30 - **Type()** = returns the type of variable
- 31 - **%s = string , %n = number , %f = float**
- 32 - You can format your string by using **string.format()**
 Example : `print("My name is {}".format("Mahmoud")) = My name is Mahmoud`
- 33 - Better way to format is : `print(f"my name is {String}") = My name is "the string`

LIST :-

- 34 - **List.append()** = add new item to the list
- 35 - **List.extend()** = will take all list from extend and put it to the list
- 36 - **List.sort(reverse="true or false")** = sort all the list by alphabet
- 37 - **List.remove()** = remove element
- 38 - **List.reverse()** = reverse all list
- 39 - **List.clear()** = remove all elements
- 40 - **List.copy()** = return shallow copy "doesn't has a relation with the original one if something happened"
- 41 - **List.count()** = count the element in list
- 42 - **List.index()** = return the index of element
- 43 - **List.insert(pos=0, element = "test")** = put element in position
- 44 - **List.pop()** = remove last element , and if you put position will remove it
- 45 - List Comprehension : **[condition for num_in_condition in iterator]**

Example : `[num*2 for num in numbers]`

And you can make it nested `[(num1,num2) for num1 in list1 for num2 in list2]`

If you want to do it with if statement :-

[output expression for iterator variable in iterable if predicate expression]

Tuples :-

- 46 - Tuples are **immutable**
- 47 - Tuple should end with **" , "** to make interpreter read it as tuple not string
- 48 - Tuples accept concatenation, **ex: a = (1,2) b = (2,4) print(a+b) = (1,2,2,4)**
- 49 - You can repeat the same tuples by multiply it with **"*"**

50 - **Tuples.count()** , **Tuples.index()**

51 - **Remember that !!** To generate the **list of tuples**, pass the call to **zip()** into **list()**.

52 - Tuple destruct meaning divide elements of tuple to variables , **ex: a=(1,2) , x,y = a print (x) = 1**
Print (y) = 2

SET :-

53 - sets not accept index and ordering

54 - To do "set" we use "{}"

55 - Set are **immutable**

56 - You can't append list into set , but you can append tuple

57 - Set elements automatic print elements as unique

Set.clear() , **Set.union() == (A|B)** , **Set.add()** , **set.copy()** "shallow" , **set.remove()** "produce error if element not in set" , **set.discard()** "will not produce error if element not in set" , **set.pop()** , **set.update()** update a set with the union of itself and others and you can put list into update parameter to update it , **set.difference()** , **set.difference_update()** will give u the difference and return to the set, **set.intersection()** , **set.intersection_update()** , **set.symmetric_difference()** will print which not in the both of 2 sets , **set.symmetric_difference_update()** , **set.issuperset()** , **set.issubset()** , **set.isdisjoint()**

Dictionary :-

Ex for it : User = {

"name" : "mahmoud",

"age" : 20

"Country" : "Egypt"

}

58 - **Dic.get()** : to access dictionary elements , **dic.keys()** will print keys like Name Age , **dic.values()** will give you all elements in the keys like mahmoud 20

And you can access it's element by calling key like : **MyDic['name of key'] = will print value**

59 - Two dimensional dictionary ex:

```
Dic {
  One {
    "name": mahmoud
  }
  Two {
    "name": ahmed
  }
}
```

60 - **Dic.clear()** to remove elements , **dic.update()** to add elements ex: **dic.update({"key" : "value"})** , **dic.copy()** create shallow copy, **dic.setdefault()** to add element in it and you can leave it without values , **dic.popitem()** to remove last element, **dic.items()** to return all elements in dictionary , **dic.fromkeys()** make a dictionary from keys stored in variable and tuples in another one

Booleans :-

61 - Any empty elements considered as False elements and True with elements

62 - Boolean operators : cond **and** cond = first condition and second one should be True ,
cond **or** cond , Any of the conditions should be True , if you put "**not**" before and , or so you make the
inverse of it

63 - **==** Equal , **!=** not Equal , **>** greater , **<** less , **>=** greater or equal , **<=** less or equal

Type Conversion :-

64 - **Str()** convert any parameter into string , **tuple()** make any list or dict or string or any variable can
be iterated into tuple , **list()** make any set or dict or string or tuple or any variable can be iterated
into list , **set()** make any dict or string or tuple or any variable can be iterated into list , **dict()** can
only make nested tuples and nested list or any variable have key and it's value can be converted
into dic() , **int()** convert any parameter into integer.

Input :-

65 - **Input()** is a method to take input from user

Control Flow :-

66 - If condition : "**do the operation**" ,elif condition : "**do another one**" , else : "**do another one**"

67 - Ternary one : **print("condition1" if Cond True else "condition2")**

68 - "**in**" operator check if the element in the variable or etc or No

LOOPS :-

69 - **While "condition is true" : "Do something"**

70 - **For "something" in "my things" : "Do something"**

71 - Code example to make relation with dictionary :

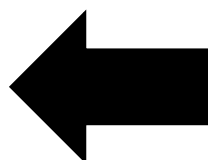
```
for skill in mySkills:
    # print(skill)
    print(f"My Progress in Lang {skill} Is: {mySkills.get(skill)}")
```

72 - **Continue** : skip iteration , **Break** : stop the programme , **Pass** : fix indentation problem

73 - **Match "Something to choice" : case 1 : do the choice 1 , case 2 : do the choice 2**

74 - **Example code for match statement :**

```
num1 = int(input())
num2 = int(input())
print("""Which operation do you want ? "
+ for addition
- for subtraction
/ for division
* for multiplication""")
choice = input()
match choice:
    case "+": print(num1+num2)
    case "-": print(num1-num2)
    case "/": print(num1/num2)
    case "*": print(num1*num2)
    case _: print("Wrong input")
```



```
5
10
Which operation do you want ? "
+ for addition
- for subtraction
/ for division
* for multiplication
50
```

LOOP ADVANCED DICTIONARY :-

For name,age in Person.items() : `print(f" your name is {name} and your age is {age})`

What about two dimension dictionary ?

75 - For main_key,min_key in Totalkeys.items() : `print(f"current main key is : {main_key} and it's value progress : ")`

For key,value in min_key.items() :
`Print(f"key is {key} and value is {value}")`

FUNCTIONS :-

76 - if you wanna to make a **function** rather than typing the same algorithm a lot , so the syntax of writing function in general is :

def MyFunction() :
`print(anything)`

77 - Your function can **has parameter or not**

78 - You can limit your parameters by writing them in **MyFunction()** , BUT! If you don't care about how many parameters that may be in your function , so write only one parameter and put before it's name **"**"** and it gonna be stored in as tuples.

79 - If you want to put a Default value for your parameters , let your parameters equal to the string **"Unknown"**

80 - The type of class MyFunction automatically is tuples , but if you want to send parameters of kind Dictionary you put **"**"** before the name of the parameter

81 - If you put **"global"** before the variable then it gonna be run for the whole code and not be scoped only in the entire method

82 - "lambda function!! = Anonymous function!!": syntax = **lambda "parameters" : "function"**

83 - Type Hinting : it's help you to now the type of parameters input ,

EX : `def MyFunc(name) -> str :` **"The '-> str' means that the type of input is strin"**
`Print(name)`

Files Handling :-

84 - To open file we use method **"open("Path","Kind")"** and you have 4 kind of parameters

85 - W = for writing , **A** = for appending , **R** = for reading , **X** = for creating file and it give error if exists
file.name return the name of it , **file.mode** return the kind of it , **file.encoding** return the type coding of the file , **file.read()** all the file , and if you put parameter integer into it , read specific number of lines , **file.readline()** read only line , **file.readlines()** return all lines in text into list , **file.write()** writing into the file , **file.truncate(parameter integer)** returns string by putting int number of the characters , **file.tell()** returns the number of characters , **file.seek()** return characters at specific index till end SOME

BUILT IN FUNCTIONS :-

- 86 - **All()** checks if elements in the iterable are true
- 87 - **Any()** checks if any element in the iterable are true
- 88 - **Bin()** convert the parameter into binary code
- 89 - **Eval()** converting expression and return result of it
- 90 - **Id()** returns his place in memory
- 91 - **Sum(iterable,start)** returns sum of numbers
- 92 - **Round(number,numofdigits)** returns the nearly of the number
- 93 - **Range(start,end,steps)** make a range of numbers from start till (end-1)
- 94 - Print details : **print("",separator,end)**
- 95 - **Abs()** return the absolute value
- 96 - **Pow(base,exp,mod)** return the pow of numbers and if you put number in mod it will return the result of the power modulus to the number you put in mod
- 97 - **Min()** return the minimum number from the set
- 98 - **Max()** return the maximum number from the set
- 99 - **Map(fun,iterator)** take a function and put it into the function which it's called map because it's map the function on every element in the iteration.
- 100 - **Filter(fun,iterator)** from it's name it take the function which will make filter operation for the iteration
- 101 - **Ord()** return the binary number of the parameter
- 102 - **Reduce(fun,iterator)** , in new versions in python reduce become in module called "functools" and it's kinda looks like "stack example" for reducing operations , converting sequence into 1 number.

Example :- `reduce(sumnumbers,[1,2,3,4,5,6]) = (((((1+2)+3)+4)+5)+6)`

- 103 - **Enumerate(iterable,start)** it's adding counter for iteration , example for it :-

List = `["mahmoud","essam","Fathy"]`

ListCounter = `enumerate(List, start = 0)`

For name in ListCounter :

Print(name)

Output in lines : `(0,"Mahmoud") (1,"Essam") (2,"Fathy")`

"FULL CODE FROM [DATA CAMP](#) EXAMPLE IN ENUMERATE :-"

```
# Create a list of strings: mutants
mutants = ['charles xavier',
            'bobby drake',
            'kurt wagner',
            'max eisenhardt',
            'kitty pryde']

# Create a list of tuples: mutant_list
```

```

mutant_list = list(enumerate(mutants))

# Print the list of tuples
print(mutant_list)

# Unpack and print the tuple pairs
for index1,value1 in enumerate(mutants):
    print(index1, value1)

# Change the start index
for index2,value2 in enumerate(mutants,start=1):
    print(index2, value2)

```

OutPut :-

```

[(0, 'charles xavier'), (1, 'bobby drake'), (2, 'kurt wagner'), (3, 'max eisenhardt'), (4, 'kitty pryde')]
0 charles xavier
1 bobby drake
2 kurt wagner
3 max eisenhardt
4 kitty pryde
1 charles xavier
2 bobby drake
3 kurt wagner
4 max eisenhardt
5 kitty pryde

```

```

2 KITTY PRYDE
4 MAX EISENHARDT

```

104 - Help(function) is used to get all informations about the function to help you

105 - Zip() : takes any number of iterables and returns a zip object that is an iterator of tuple
And taking from every list you put element in every list ,”meaning is list1[0],list2[0],list3[0] ,
List1[1],list2[1],list3[1] and etc ...

“THIS EXAMPLE FROM DATA CAMP TO MAKE IT EASIER TO UNDERSTAND ZIP”

```

mutants = ['charles xavier', 'bobby drake',
            'kurt wagner', 'max eisenhardt', 'kitty pride']
aliases = ['prof x', 'iceman', 'nightcrawler', 'magneto', 'shadowcat']
powers = ['telepathy', 'thermokinesis',
          'teleportation', 'magnetokinesis', 'intangibility']

# Create a zip object using the three lists: mutant_zip
mutant_zip = zip(mutants, aliases, powers)

# Print the zip object
print(mutant_zip)

# Unpack the zip object and print the tuple values
for value1, value2, value3 in mutant_zip:
    print(value1, value2, value3)

```


106 - **Reversed()** from it's name reversing the iterations from "0 to 10" to "10 to 0"

Iterators and iterables :-

107 - **Iterables** are objects can be looped and get it's own items , **"STRING - LIST - SET - DIC - TUPLE"**

108 - **Iterators** are objects make **non-iterable into iterable**

109 - to make iterator into iterable just write **iter(object)** and it gonna be iterable and we use **next()** method it iterate char and char and char ... till we reach **StopIteration**

Generators :-

110 - we can say that generator work as use as functions without any difference, just function use keyword **"return"**, generator use **"yield"** but the big difference that we use **"next method"** in generator and that because Memory is saved as the items are produced when required, unlike normal Python functions

Decorators(meta programming) :-

111 - decorator from it's name changing the design or something in the function and return it with different style

it's syntax :-

```
def MyDecorator (function) :
```

```
    def MyNestedOne() :
```

```
        "style"
```

```
        Function()
```

```
        "style"
```

Return Function

You can use something called **"sugar syntax"** by writing **@MyDecorator** to the line before the function you want to decorate

Errors and Exception Handling :

112 - Try handle and it's except is here to survive you from the whole error coding by using only

Try : "the code u want to do"

except : "the error you want to print"

113 - After execution of error normally the code gonna stop and will return you the exception

114 - **Finally** : used after the code done even if the error exist will perform something

Try : "the code u want to do"

except : "the error you want to print"

Finally : "The code you want to run even the main code exist error"

115 - **Except "without anything"** : means code will print in general that there's error

116 - **Except "specific error"** : means will exist this exception if specific error has happened

Debugger :-

117 - **Debugger** is one of most effective ways to make you up with the every line of the code and And the programme will give you all details happens in every line till you reach to the error

"This will help you with large codes"

118 - **BreakPoints** : are points which your gut instinct may find error in this line of the code

Regular Expression :-

119 - **A Regular Expression (or Regex)** is a pattern (or filter) that describes a set of strings that matches the pattern

120 - To test the Regex I use **"Pythex: a Python regular expression editor"**

Or **"regex101: build, test, and debug regex"**

121 - And this sheet cheat for it : **"PCRE Regular Expression Cheatsheet - Debuggex"**

OOP:-

122 - Sure at least you have faced a programming language like C++ or C or java and know about what Is Object oriented programming, anyway we are gonna talk about it with python this time

123 - OOP : is a **programming paradigm** based on the concept of "objects", which can contain **data** and **code**.

124 - data in the form of fields (often known as **attributes** or **properties**), and code in the form of **procedures** (often known as **methods**)

125 - to start with OOP, Your first step is defining **class** , so you make new python file and start is with write **"CLASS"**

126 - **Remember that!** Any method with **__** before it and **__** after it called **Dunder or magic method**.

127 - To define "Constructor" or "Initializer" for the class we write : **def __init__(self):**

128 - Wait...What's the SELF ?? ok self refers to the current instance created from the class and "Self" must be the first parameter

129 - It can be any word rather than "self" but it's important to put word refers to the current object.

130 - In other languages to define new object we use "New" to allocate it in memory, In python we don't need for "new" to define it

THIS IS EXAMPLE FOR CLASS :-

Class ex1 :

Def __init__(self):

Print("Hello World")

"Main class"

Variable1 = ex1() #now you can access ex1 methods and u made ur first object from it

131 - This is another example will make it easier to be familiar with it :

Def person:

Def __init__(self,Name,Age,Address,gender):

```
Self.Name = Name
Self.Age = Age
Self.Address = Address
Self.gender = gender
```

```
Def Details(self):
```

```
Return f"{self.Name} {self.Age} {self.address} {self.gender}"
```

"OUR MAIN CLASS" :

```
X = person("Mahmoud", "20", "MyAddress", "Male")
```

```
Print(X.Details())
```

132 - It's so easy right ? **self** is the object created, **self.name** is the attribute which will store the name of the person , **self.age** will store the **Age** of the person and **address** the same way

133 - X.Details() it's very simple, will access to the method you built and will return the data written in the print method

134 - **Class method** X **instance method** : difference between them that class doesn't need for object to get called and no need for 'self' parameter

135 - So what is the parameter we use ? that name of it is 'CLS' refer to the class

136 - To def a class method :

```
@classmethod
```

```
Def cmethod(cls):
```

```
Print(cls.attribute)
```

137 - **Class method** only used to do something with the class itself

138 - "**STATIC METHOD**" : no need for parameters in general, and you can call it in main easily without needing to make object of it

139 - And this the big difference between static and class method :

140 - After ending your oop class, how to do a long print for all my My methods ? we define method called "**__str__**" like other langs Which create methods called "**ToString**"

141 - There's another magic method called "**__len__**" used for return length of the container , And it called win we use "len" built in function

142 - **Inheritance** : it's one of the 4 important concepts in OOP, it's allow for the user to take all elements from the "Base class" and give it to another class "Child one"

143 - The syntax to inherit : **Def MyBaseClass() : #this is base one, Def MyChild(MyBaseClass) : #Now inheritance between to of them done from my base to child** , but there's some notes we should take it when we do inheritance.

144 - From the meaning of **inheritance** you got all attributes from **MyBaseClass** right ? but you didn't access to his attributes totally, so to not repeat the same methods again and do the same in **__init__** of the child, We write the name of the class and **__init__** and the attributes like this way :

```
def MyChild(MyBase) :
```

Class Method	Static Method
The class method takes cls (class) as first argument.	The static method does not take any specific parameter.
Class method can access and modify the class state.	Static Method cannot access or modify the class state.
The class method takes the class as parameter to know about the state of that class.	Static methods do not know about class state. These methods are used to do some utility tasks by taking some parameters.
@classmethod decorator is used here.	@staticmethod decorator is used here.

```
def __init__(self):
    MyBase.__init__(self,other attributes)
    "your init attributes"
```

145 - This way in java we call it "Super()" which lead us to inherit the constructor of base too.

146 - And for your happiness you can use it in python also !! with the following syntax :

```
def MyChild(MyBase):
    def __init__(self,attributes):
        Super().__init__(self,other attributes)
        "your init attributes"
```

147 - Does python accept multiply inheritance ? YES!!, there's languages doesn't accept it like java, so python solved this problem with this syntax : `def MyChild(MyBase1,MyBase2)` : "Now you inherited 2 of base classes into 1)

148 - **Polymorphism** : it's one of the 4 concepts of OOP means "**Many Forms**" which you can use the same function but in other way of use "**Which we say it override method with same name but difference In output**"

149 - Here's little Story may let it easy for your mind to get : **A boy starts LOVE with the word FRIENDSHIP, but girl ends LOVE with the same word FRIENDSHIP. Word is the same but attitude is different, This beautiful concept of OOP is Polymorphism**

150 - **Encapsulation** : it's one of the 4 concepts of OOP which is the privacy control of variables you define in the class to access to them.

151 - The popular kind of accessing are 3 : **Public < Protected < Private**

152 - Public access "**self.name = name**" give it general access for all classes and can modify it "**No signs**"

153 - Protected access "**self._name = name**" give it general access for only within same class "**one _**"

154 - Private access "**self.__name = name**" doesn't give access for it and you can't change it "**two __**"

155 - for **sad** , as there's no keywords for accessing in python , you also can access the attributes even if it's private "**But no one can now it except the workers of it**" : **self._className__PrivateVariable**

156 - sure you won't call the name for every instance and change it "**If all of them public**" , remember that you can't access **private attributes** , How to solve it ??

157 - Getters and Setters : setters are method to assign the values , and getters to return it

158 - It's so easy ! , syntax is : **def set_name(self,NewName) : self.__name=NewName ,**
def get_name(self) : return self.__name , that's it ! this will solve the private accessing problem

159 - Old way to get it right ? let's talk about **@property decorator** with Python !

160 - **@property** : used to give "special" functionality to certain methods to make them act as getters, setters

161 - This code will make it easier for you to understand it using :

```
class x:
    def __init__(self):
        self.__name = ""
        self.__age = 0
```

```

# OLD WAY
def set_name(self, NewName):
    self.__name = NewName

def get_name(self):
    return self.__name

# NEW WAY
@property
def name(self):
    return self.__name

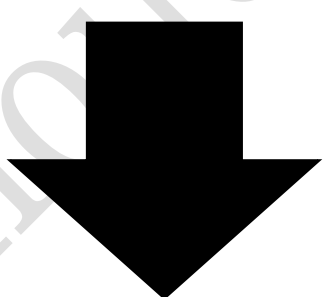
@name.setter
def name(self, NewName):
    self.__name = NewName

#OLD :
user1 = x()
user1.set_name("mahmoud")
print(user1.get_name())
#NEW :
user2 = x()
user2.name = "mahmoud"
print(user2.name)

```

162 - Abstract Class : the last concept of the 4 concepts OOP , You can say that abstract is definer for your methods, That's all !

163 - This syntax code to explain the abstract method :



```

from abc import ABCMeta, abstractmethod # abc used to
import abstract definitions
class x(metaclass=ABCMeta): # now you have defined it as
    abstract class
# As we said abstrct class is a definer, compiler will
force you to implement all his methods
# Abstract class can have abstract method or solid method
    @abstractmethod
    def hello(self): #This one is abstract
        pass

    def world(self): #this one is solid
        pass

class y(x):
    def hello(self):
        print("Hello")

```

```
def world(self):
    print("World")

# some notes :
# you can inherit method to another class, but you can't
# make instance from abstract class! will cause error
# if method is abstract you should call it, if solid you
# can call it or not as you want
```

164 - Doc String : when you print the **help(function)** it returns for you all details about the package With parameters and etc.

To make Doc String :-

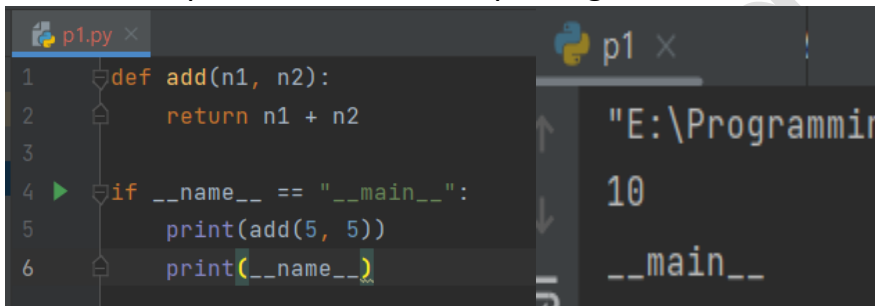
```
def MyFunc() :
    """ THAT'S DOC STRING """
```

Now to print it : **print(MyFunc.__doc__)**

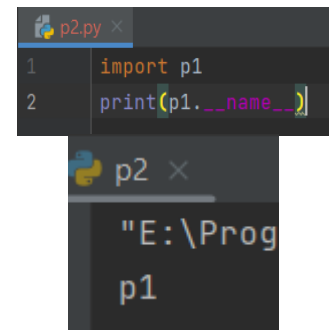
165 - __name__ = __main__ : runs blocks of code only when our Python script is being executed directly from a user.

166 - this topic has a lot of ways to use, **1-** it's only executed in the main script not in the imported class , **2-** in general your class name is **__main__** so by using it you give it name when get imported

167 - here's example for code then importing it to another class :



```
1 def add(n1, n2):
2     return n1 + n2
3
4 if __name__ == "__main__":
5     print(add(5, 5))
6     print(__name__)
```



```
1 import p1
2 print(p1.__name__)
```

168 - Now you have done all OOP Notes you may need ❤️

DATA BASE :-

169 - Database where you can store your data and I think you have at least good knowledge about it

170 - “to access data base you can use **SQLite** and coding details will start in second point”

171 - To import SQL in python we use **“import sqlite3”**

172 - the full code imported to GITHUB and the next syntax will explain all details to how use it “remember to download SQLite”

the reference of database was by “Eng.Osama Elzero”



```
# here will discuss SQL with python "DATA BASE"
# to import SQL in python will use the next syntax
import sqlite3
db = sqlite3.connect("Example.db") # to create database and connect to it
db.execute("CREATE TABLE if not exists 'person' (name text,age
integer,user_id integer)") # create the table and fields
Cursor = db.cursor() # to create cursor
#Cursor : will handle all operations with sql
#you can create table as use as execute data
#INSERTING DATA INTO DATABASE :
Cursor.execute("INSERT INTO 'person' (name,age,user_id) VALUES
('mahmoud',20,1)")
#UPDATE DATA INTO DATABASE :
Cursor.execute("UPDATE person set name = 'name1' where name = 'name2' ")
#DELETE DATA FROM DATABASE
Cursor.execute("DELETE from person where name = 'your name'")
#RETURN DATA FROM DATABASE
Cursor.fetchone() #will return single row of the database "like next in
iterable" till return none
Cursor.fetchall() #will return all rows of the database as list of tuples
Cursor.fetchmany("specific number of rows you want") #will return specific
number of rows from the database
#COMMIT THE ORDER AND CLOSE IT
db.commit() # to do the order and save it into the database
db.close() # To close the DataBase
#to protect your self from database injection "don't use FORMAT way"
```

173 - by this code “which is not enough to talk about DBMS” you can connect python with it

174 - my recommendation for you to take this course which will help you a lot in data’s life

Modules :-

175 - To import module it’s really easy , Just say “**import module**”

176 - And if you want from the whole module only function , just say “**from module import fun**”

177 - **Random** : is a module generate random numbers

178 - **Sys** : is a module for your system and it has function to get path and append it “for now”

179 - Creating your module : it’s so easy , make your python file and put it into the path where you can import your modules from it , and done import it into your file!

180 - **Package** : is amount of modules , module which we call it

181 - We can download packages by using **PIP** , for external modules : [PyPI · The Python Package Index](https://pypi.org/)

182 - If you want to download package just write : **pip install “package name”** or you can just write the package name in the complier and it will give you error to download it

183 - To know the methods into module , write “**dir(module)**”

All modules that I could find at the time of uploading this file will upload it into python file and if I met any module impressive will upload it into this repository :

<https://github.com/MahmoudEssam707/Python-Modules> (github.com)