# home

revise map and reduce implementation

#website to learn python->data-flair python-->https://data-flair.training/blogs/python-tutorials-home/

# file names and file path operations and other file related operations:

-filepath in windows are seperated by \

whereas filepath in mac os and linux are seperated by /

so inorder to generate a script that can traverse over different operating systems independantly we can use os module.

if the file named sample.txt is in folder 1.3 which is in folder 1.2 which is in folder 1 which is in c drive

then by simply using

path=os.path.join(folder1,folder1.2,folder1.3,sample.txt)

print(os.sep) #explains what type of seperator does the os uses

os.getcwd() #will return the current working directory in string format

if you give any folder or file without any path then the interpreter assumes that the file is in current working directory, it choses to analyze relative path (cwd does not have any relation to the location of interpreter or compiler).

os.chdir(path) can be used to change the cwd to the required path

paths can be classified as two types:

1:absolute file path: this file path is configured right from the root folder and it stays the same even if the cwd changes

2:relative file path: this file path is in relation to cwd and if it changes the path also should change inorder to be appropriate

in relative file path we have two conventions:

. -> current directory or current folder

.. ->parent folder

. is are used when we know what are the files that are present in current folder i.e, for example i know c: will have users folder inside

ex: ./python/sample.py

here in the current directory it will check for python folder and then it looks for a file named sample with py extension

.. is used when we want to move to parent folder and traverse from there to another path, it is done like this cd ../ will change cwd to parent directory

another example is ..\sample\vini.txt what happens here is from the current directory it will move to parent folder and then opens folder named sample and then opens a file named vini

below are some of the methods we use in path traversing

os.path.abspath(<filename or folder name or relative path>) # the output will be the cwd+<filename or foldername> this is useful when we have to create a folder or file in the cwd

similarly we can also create folder or path in the parent or grandparent directory:

creating file or folder in parent directory: os.path.abspath('../<file or folder>')

creating file or folder in grandparent directory: os.path.abspath('../../<file or folder>')

-os.path.isabs(path): this will return true if the path passed is an absolute path else it will return false, it will return True if the given absolute path does not exist in real

-print(os.path.relpath(r'C:\Users\saivinil.pratap','c:\\Users')) #output:saivinil.pratap

this will print the relative path from the second parameter and will not throw error if the second parameter is not a parent or grandparent or so, it will just return the maximum match possible

-os.rename(old\_name(withpath),new\_name(withpath)

ex:print(os.rename(r"C:\Users\saivinil.pratap\Desktop\teams online.py",r"C:\Users\saivinil.pratap\Desktop\teamonline.py"))

-os.path.dirname(path)# this will return the **parent folder absolute path** of the last folder

Ex: print(os.path.dirname(r'C:\Users\saivinil.pratap\jupyter useful notebooks\abc\def\ghi.txt'))

# output: C:\Users\saivinil.pratap\jupyter useful notebooks\abc\def

-os.path.basename(path) # this will return the last file or folder (if file is not passed only folder path is passed) of the path passed

-os.path.exists(path) # will return True if the path exists in real in the os , else false

-os.path.isfile(path) # will return True if the path passed is a file , else false #it does not just decide merely by checking if the last part of the path has an extension or not, even if you pass a file which does not have any extension, then also it returns True

-os.path.isdir(path) #will return True if the path passed ends with a folder, not a file, else false

isdir and isfile return true only if the **file or directory exists in real**, else false

-os.path.getsize(path) #will return the size of file or folder in bytes

-os.listdir(path) will return all files and paths under the current path, will raise an error if path doesnot exist in real, **does not go recursive.**

-os.makedirs(path) will create folders , the input path passed must be real and it can be absolute filepath or relative filepath

the root folder is the starting folder or the lowest folder, in windows it is C:\\ in mac or linux it is /

--copy and rename commands

import shutil

shutil.copy(filepath,directory) #this will copy the file from filepath to specified directory

shutil.copy("C:\Users\saivinil.pratap\Desktop\python\_virtual\_env\pyvenv.cfg",C:\\Users\\saivinil.pratap\\Desktop\\vinil.txt) #this will copy the pyvenv file from current file path and it will paste with a new name called vinil in desktop

shutil.copytree(source\_path,destination\_path) #this will recursively copy all folders from source path untill end into destination\_path, if destination\_path does not exists , then it will create and then perform copy operations

shutil.move(file/folder path,destination\_path)#this will recursively move all folders and files from specified source to destination path if destination\_path does not exists , then it will create and then perform copy operations

shutil.move(file path, destination\_path)

ex:shutil.move(r"C:\Users\saivinil.pratap\Desktop\python\_virtual\_env\pyvenv.cfg",r"C:\Users\saivinil.pratap\Desktop\python\_virtual\_env\pyvenvs.cfg")

# this will move file from source to destination path and if you declare destination path with extension newfilename.txt, ex:r'D:\b\name.txt' this is technically renaming the file as the contents from old file are moved to a new file with different name

#can also be used to rename it by specifying the same directory path,but with a different file name

ex:shutil.move(r"C:\Users\saivinil.pratap\Desktop\python\_virtual\_env\Scripts\python automation scripts\vpn.py",r"C:\Users\saivinil.pratap\Desktop\python\_virtual\_env\Scripts\python automation scripts\vpncopy.py")

--temporary and permanently delete commands:

os.unlink(r"C:\Users\saivinil.pratap\Desktop\sample\_folder\Account.class")#this will **permanently** remove the file from the memory,check if it can delete a directory (with files inside it) as well

os.rmdir(r"D:\b")# will delete the folder in the path **permanently** will work only if the directory is empty

shutil.rmtree(r'D:\b') #will start deleting **permanently** the folder and its contents recursively including b folder

# you can avoid this by performing a dry run where you comment the delete command try first printing the paths and then checking them and uncomment and carry on..

import send2trash

send2trash.send2trash(r"C:\Users\saivinil.pratap\Desktop\Capstone Project")#this will **move the file or folder from its location to recycle bin** instead of deleting permanently

-to traverse over a directory:

path=r"D:\A"

for folder\_name,sub\_folders,file\_names in os.walk(path):

'''

this is just to navigate the current folder and go till the very end,

this is helpful when you want to check many folders which have different types of files and folders

you can use regex or other function to find particular set of files(with particular extensions) or folders

you can use **deleting syntaxes such as unlink or copying syntax such as copytree or any other command of your choice to delete or move files to a new locations** in the below loop

'''

print('the folder is '+folder\_name)

print('the subfolders in the '+folder\_name+'is '+str(sub\_folders))

print('the filenames in the '+folder\_name+'is '+str(file\_names))

print()

--in python we can open a file in 3 modes

f=open(file,mode)

3 modes(all extended modes will have basic conditions, example w+ will create a file if doesn't exist and):

1:r -> opens a file for reading, it is default mode ,cursor is placed at starting

2:w -> opens a file for writing, if a file is already existing it's content is deleted and writing starts again, if not creates a new file with the filename provided

3:x -> opens a file for e'X'clusive creation i.e, creates a file if it doesn't exist, else throws an error

4:a -> opens a file for Appending data, appends data at the end of the file,creates a new file if it does not exist

5:t -> opens a file in text mode, it is default mode

6:b -> opens a file in binary mode, used for scientific computations

7:rb -> opens a file to read in binary mode

8:r+ -> opens a file to read and write mode,file pointer is placed at beginning of the file

9:rb+ -> opens a file in reading and writing in binary format,file pointer is placed at beginning of the file

10:w+ -> opens a file for writing and reading ,creates a file if doesn't exist and erases the existing content if file exist

+ -> open a file for updating(reading/writing)

wb+,ab,a+,ab+ are some other formats

--file\_var=open(r"path","mode") now file\_var variable is an object

here are some of the operations performed on file\_var file object:

file\_var.mode-> returns the mode of the file

file\_var.closed-> returns true if the file\_var is closed

file\_var.name-> returns the name of the file

file\_var.softspace-> a read-write attribute that is used internally by the print statement to keep track of its own state. A file object does not alter nor interpret softspace in any way: it just lets the attribute be freely read and written, and print takes care of the rest. #need clarity

--close a file after opening it else, it will be stored in ram

--safer way of opening a file is below

try:

f=open("path","mode",encoding='encoding\_format if it is different')

finally:

f.close()

encoding

--#to get the encoding format of a file

f=open(r"C:\Users\vinil\OneDrive\Desktop\notes.txt","rb").read()

import chardet

result=chardet.detect(f)

encod\_f=result['encoding']

encod\_f

--when a file is opened in read mode,

f.read() will read all the content

f.read(digit) will read from 0 position to digit-1 position,next f.read(digit) will give the data after the updated cursor at digit-1 position #need clarity

f.tell()-will return the cursor position

f.seek(position) - will move cursor to specified point

for i in file\_var:

print(i)#prints data line by line

f.read\_Lines() - will return list of lines from the file

### to get the path of the current file inside the same file

check the below code along with the comments

from pathlib import Path

print(Path(\_\_file\_\_)) # will give the name of the file

print(Path(\_\_file\_\_).resolve()) # will give the ABSOLUTE path of the file

print(Path(\_\_file\_\_).resolve().parent)# will give the ABSOLUTE path of the parent directory

print(Path(\_\_file\_\_).resolve().parent.parent)# will give the ABSOLUTE path of the grand parent directory

os.path.join(BASE\_DIR, 'discount\_web\_api')

sys.path.append(BASE\_DIR.\_\_str\_\_())

#### important note

the Path(\_\_file\_\_) and Path(\_\_file\_\_).resolve() will return pathlib.PosixPath object which works well with os.path.join and can be passed as a direct parameter ex: os.path.join(BASE\_DIR, 'discount\_web\_api') but to work with sys.path you need to call the dunder \_\_str\_\_ method ex: sys.path.append(BASE\_DIR.\_\_str\_\_())

# scope explanation:

--global scope is the area which is outside all functions, the variables declared here are global variables

--local scope - local space of a function, which contains its code and, the variables declared here are local variables

--code in global scope can't use any local variables

--local scope means the assignments done to a variable or anything is limited to the function in which it is defined, global scope means the assignments done to a variable or anything is accessible from anywhere of the program(even in other functions) if a function has to access an variable but if the variable is not defined inside the function then the interpreter looks if there is any global variable that is existing with the same name already(assigned before the current function is called), if so that variable will be used.

--code in one local function cannot use local scope variables of other functions

-- if we want to assign a new value to a global variable which is already assigned a value, to change it inside a function we have to use "global" keyword, if you just want to use a global variable you can use it directly the same way you use local variables, if you use **global value inside a local function** by calling **global** keyword, then the ultimate reassigned (or manipulated value) after the function executes will be used. Ex: in the below snippet, after the function call spam() is executed g will be assigned a string, and is available globally as a string (execute the below function to get clear idea)

ex:

g=5

def spam():

global g

print("before changing global variable it's value is :",g)

g='the global value is changed inside the function'

print(g)

spam()

print(g)

--a scope can be thought as an area of source code and a container of variables

--sys.exit(value or string)

exit() function allows the developer to exit from Python. The exit function takes an optional argument, typically an integer, that gives an exit status.

The optional argument arg can be an integer giving the exit status (defaulting to zero), or another type of object. If it is an integer, zero is considered “successful termination” and any nonzero value is considered “abnormal termination” by shells and the like. Most systems require it to be in the range 0–127, and produce undefined results otherwise. Some systems have a convention for assigning specific meanings to specific exit codes, but these are generally underdeveloped; Unix programs generally use 2 for command line syntax errors and 1 for all other kind of errors. If another type of object is passed, **None is equivalent to passing zero**, and any other object is printed to stderr and results in an exit code of 1. In particular, sys.exit("some error message") is a quick way to exit a program when an error occurs.

Since exit() ultimately “only” raises an exception, it will only exit the process when called from the main thread, and the exception is not intercepted.

--Compile-time is the instance where the code you entered is converted to executable code while Run-time is the instance where the executable code is running

--%timeit - will give average iteration time it will take for one iteration

----a='{}{}'.format(10,20)

print(a)#o/p:1020

--a=2

b=10

b\*\*=a

b#o/p:100

--print('{name} inital {surname}'.format(name='vinil',surname='pratap'))#o/p:vinil inital pratap

-- \*\*=, &=, |=, ^= are some unknown assignment operators

--unzip <folder\_name> will unzip the folder in putty

--python <file\_name.ext> install -will install the filein putty.

--after running setup.py file any package will be installed.

--one package after installation maybe depended upon the other packages which further needs to be installed.

--cython package is used to get c features in python,It makes writing C extensions for Python as easy as Python itself.Cython is a programming language that aims to be a superset of the Python programming language, designed to give C-like performance with code that is written mostly in Python with optional additional C-inspired syntax.

--In programming, a metasyntactic (which derives from meta and syntax ) variable is a variable (a changeable value) that is used to temporarily represent a function .

"Foo" and "bar" as metasyntactic variables

--Python provides, you can use the PyPI (Python Package Index). It is a repository of third-party Python modules

--df[temp\_col]="" used to create a column on a dataframe df with no values

--Let’s now talk about Python architecture and its usual flow –

i. Parser

It uses the source code to generate an abstract syntax tree.

ii. Compiler

It turns the abstract syntax tree into Python bytecode.

iii. Interpreter

It executes the code line by line in a REPL (Read-Evaluate-Print-Loop) fashion.

# arithmetic operators:

a//b will return integer divison

a/b will return float divison

a^b mod m can be calculated simply using pow(a,b,m) this pow is not a module from math library it is a general function and it does not need import statement

ex:

x = 7

y = 2

z = 5

print(pow(x, y, z)) # 4

# oops\_explanation\_with\_python:

--Major principles of object-oriented programming system are given below.

Class

Object

Method

Encapsulation

Polymorphism

Abstraction

Inheritance

(first letters of above principles are COMEPAI)

MAIN PILLARS IN OOPS CONCEPTS ARE:(EPAI)

E-ENCAPSULATION

P-POLYMORPHISM

A-ABSTRACTION

I-INHERITANCE

## Class

Class is the blue print based on which objects are created

Syntax:

Class <class name>():

//optional method

def \_\_init\_\_(self,<any parameters that you wanna use>):

//parameter initialization

//methods

Note: class name should follow camel casing (each word should start with a capital letter) ex: SportsCar

--creating a class in python:

class Parrot:

#notice below how a class attribute is being declared, similar to attributes being declared in methods are called method attributes, attributes declared in a class are called class attributes

species = "bird"

# instance attribute

def \_\_init\_\_(self, name, age): #here \_\_init\_\_ is a special method, also known as instantiation method (which is used to initialize the variables(as it is called automatically) this is similar to constructor concept in java and cpp) and any variable surrounded by a double underscore is called a special variable, and any method which is surrounded by double underscores is called special method, special methods are also called as magic methods (because they help override the functionality of builtin functions or custom classes ) or dunder methods (because they are surrounded by double underscore)

self.name = name #in the above declaration we had defined name and age as attributes but unless it get assigned to the object using the self.name it will not get assigned to the object.

self.age = age

# instantiate the Parrot class

blu = Parrot("Blu", 10)

woo = Parrot("Woo", 15)

# access the class attributes

print("Blu is a {}".format(blu.\_\_class\_\_.species)) #or print("Blu is a {}".format(Parrot('','').\_\_class\_\_.species)) to fetch a "class attribute"

we can fetch class attribute using object or class name

1. using object

you have to use object.\_\_class\_\_.<attributename> and name,age are called "instance attributes"

print("Woo is also a {}".format(woo.\_\_class\_\_.species))

1. using classname

print(Parrot.species)

# access the instance attributes

print("{} is {} years old".format( blu.name, blu.age))

print("{} is {} years old".format( woo.name, woo.age))

--class inside a class:

class Student:

def \_\_init\_\_(self,name,age):

self.student\_name=name

self.student\_age=age

self.lap=self.Laptop('lenovo','i5',9)#declaring the object of the class which is defined in the current class, notice the self getting declared on both sides

def show(self):

print(self.student\_name,self.student\_age)

self.lap.show()

class Laptop: #declaring class inside a class

def \_\_init\_\_(self,brand,cpu,ram):

self.brand=brand

self.cpu=cpu

self.ram=ram

def show(self):

print(self.brand,self.cpu,self.ram)

s1=Student('sai',6)

s1.show()

lap1=Student.Laptop('hp','i6',8)#creating an object of laptop using Student class

lap1=s1.Laptop('hp','i6',8)#creating an object of laptop using Student object.

Student.Laptop('hp','i6',8).show()#calling an method of laptop using laptop object and Student class

lap1.show()

## object

object has 2 charecteristics:

1. attributes- (what a object has)it explains the physical appearance or properties of object using metrics ex: each parrot might have different color,height and name
   1. variables are of two types:
      1. instance variables: variables which change with the change in object are called instance variables in simple words, we can say variables declares inside the init method are called instance variables. for example,name variable differsn for each object created through person class
      2. static variables: variables which are common for all the objects (what a object has),that are declared outside init and other methods are called static variables, for example wheels variable for all the car objects remain the same, and also it acts as a single point so that when we change the value here and execute it from then it changes for all the objects it is instantiated through, whereas the color of each car may differ , so the color is better when declared as instance variable.
2. behaviour- (what a object can do)it explains what the object can do , ex: a parrot can sing or fly

the concept of OOP in python is to create reusable code,AKA DRY(Dont Repeat yourself)

class: it is a blue print for the object, it contains all details of what a object should have.

object: it is an instantation of class,when an object is instantiated , memory is also allocated for it.

in short:

every object KNOWS SOMETHING(attributes), and it DOES SOMETHING(behaviour), attributes are stored in variables and behaviours are stored in methods.

classname.method(object) is same as object.method() #both does the same job of calling a method using an object, method() is defined as

def method(self):

//statements

obj.method() will be changed as method(obj) internally and when it goes to the function definition method(self) the self will get replaced by object and execution continues

everything in python is an object and every object needs some memory and it is stored in **heap Memory**,size of an object depends upon number of variables and size of the variables it has.

## methods

methods will have a set of statements which are mentioned with a name (function name or method name) name of the method/function should follow snake casing (a naming convention where set of words or letters separated by a underscore)

def <method name>(<arguments (optional)>):

“”” doc string””

//statements

--defining methods in class:

class Parrot:

# instance attributes

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

# instance method, an instance method will definetly have self as a parameter

def sing(self, song):

return "{} sings {}".format(self.name, song)

def dance(self):

return "{} is now dancing".format(self.name)

# instantiate the object

blu = Parrot("Blu", 10)

# "sing and dance are instance methods"

print(blu.sing("'Happy'"))

print(blu.dance()) #**here the () implies we are running or executing the method, if you use the () then the interpreter returns the address of the object where the data that is to be returned will be stored**

#last stopped point

## inheritance

types of inheritance:

single- A->B

multiple a,b->c

multi level A->b->C

hybrid- a mix of two or more of the above inheritance types is an example of hybrid inheritance.

how init works in inheritance:

if a \_\_init\_\_(self) method is in a class, then it will be called instantly without any explicit call, if class b is inherited from class a (a->b) and class b dont have an init method and class a has an init method and if an object is called from class b, then as the class b does not have an init method an init method from class a is called (as class a is the parent of class b),if the class b (which is inherited from the class a) is also having the init method and for suppose you want to call the init method in the parent class a then you can call using the following syntax- super().\_\_init\_\_(),

Now comes to the interesting part of MRO-Method Resolution object: if there is a multiple inheritance and in the child class if we try calling method from the parent class then the firstly inherited class method is called, observe the below example:

class A:

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

print('inside init of A')

class B:

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

print('inside init of B')

class C(B,A):

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

print('inside init of C')

super().\_\_init\_\_()

c=C()

output:

inside init of C

inside init of B

#here init of class B is called because B is getting inherited first in class C(B,A)

# parent class

class Bird:

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

print("Bird is ready")

def whoisThis(self):

print("Bird")

def swim(self):

print("Swim faster")

# child class

class Penguin(Bird):

birdfamily='arthropoda'

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

# call super() function

super().\_\_init\_\_()

print("Penguin is ready")

def whoisThis(self):

print("Penguin")

def run(self):#all methods other than static and class method are instance methods,they are of two types and are discussed below

print("Run faster")

@classmethod

def bird\_family(cls):

return cls.birdfamily #Penguin.bird\_family() this is the way to call class method, classname.classmethod() here we use cls instead of self as parameter

@staticmethod# a static method will not have any parameters.

def basic\_info():

return "this class is about birds and their capabilities."

peggy = Penguin()

peggy.whoisThis()

peggy.swim()

peggy.run()

Penguin.bird\_family()#you can call a class method using class name

Penguin.basic\_info()#you can also call a static method using class name

three types of methods in OOPS python(unlike variables which are only two types-static variables,instance variables):

1:instance methods- the methods which use self as parameter are called instance methods

further classified into 2 types:

1.1 accessor methods- this methods are used to access the variables ex: get methods

1.2 mutator methods- this method is used to mutate the varibles ex: set methods

2:class methods

class methods use class variables it uses decorator @classmethod to differentiate the method from an instance method. the above bird\_family method is an example of class method

3:static methods

static methods are methods which dont have the need of class variables or static variables and it doesnt have any parameters passes to it (unlike self,cls), check the above example method basic\_info, a decorator @staticmethod is needed to make the interpreter understand

notes:In the above program, we created two classes i.e. Bird (parent class) and Penguin (child class). The child class inherits the functions of parent class. We can see this from the swim() method.

Again, the child class modified the behavior of the parent class. We can see this from the whoisThis() method. Furthermore, we extend the functions of the parent class, by creating a new run() method.

Additionally, we use the super() function inside the \_\_init\_\_() method. This allows us to run the \_\_init\_\_() method of the parent class inside the child class.

## polymorphism

a concept in object oriented programming where a common interface is provided to multiple forms(data types)

Suppose, we need to color a shape, there are multiple shape options (rectangle, square, circle). However we could use the same method to color any shape. This concept is called Polymorphism.

class Parrot:

def fly(self):

print("Parrot can fly")

def swim(self):

print("Parrot can't swim")

class Penguin:

def fly(self):

print("Penguin can't fly")

def swim(self):

print("Penguin can swim")

# common interface

def flying\_test(bird):

bird.fly()

#or you can also use class for the above example

class flying\_tests():

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

clas.fly()

#instantiate objects

blu = Parrot()

peggy = Penguin()

# passing the object

flying\_test(blu)

flying\_tests(blu)

flying\_test(peggy)

note:"Polymorphism allows the same interface for different objects, so programmers can write efficient code."

In the above program, we defined two classes Parrot and Penguin. Each of them have a common fly() method. However, their functions are different.

To use polymorphism, we created a common interface i.e flying\_test() function that takes any object and calls the object's fly() method. Thus, when we passed the blu and peggy objects in the flying\_test() function, it ran effectively.

## Encapsulation

Encapsulation means **restricting the direct access to some components of an object**

**Encapsulation hides internal working so that you can change it later, hides details at implementation level.**

**Hides internal working of objects from outside world**

**Encapsulation means making it as a capsule, what does a capsule do? It takes all of required elements and make it as one i.e., simplifying behaviour** , **also from outside a capsule looks as a one entity by hiding its inside contents i.e., hiding data**

Therefore encapsulation does

1. simplifying behaviour
2. hiding data

in python we can restrict the access of methods and variables and thereby prevent direct modification of them.private attributes are declared using prefix - single underscore \_ or double underscore\_\_.

ex prgrm:

class Computer:

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

self.\_\_maxprice = 900

def sell(self):

print("Selling Price: {}".format(self.\_\_maxprice))

def setMaxPrice(self, price):

self.\_\_maxprice = price

c = Computer()

c.sell()

# change the price

c.\_\_maxprice = 1000

c.sell()

# using setter function

c.setMaxPrice(1000)

c.sell()

note:We used \_\_init\_\_() method to set the maximum selling price of Computer. We tried to modify the price. However, we can't change it because Python treats the \_\_maxprice as private attributes.

As shown, to change the value, we have to use a setter function i.e setMaxPrice() which takes price as a parameter.

## abstraction

Data Abstraction: often abstraction and encapsulation are treated as synonyms, and it is true to some extent because abstraction is achieved through encapsulation. abstraction hides the implementation behind and allows to call the function, abstracting something means giving name to the functionality based on what the function or program does at core.

An example for abstraction: when it comes to driving a bike, you know how to drive it, viz., applying breaks, turning the vehicle on, changing gears and raising speed, but you don’t know its implementation. In the same way abstraction will help you call a function(ex: finding whether a number is prime or not) and get the work done, but there is no need for you to know the logic of the function. Similarly hiding the logic and providing the implementation is abstraction

**Abstraction hides complexity by giving a more abstract(clearer) picture, abstraction hides details at design level**

**Focusses on relevant information by hiding unnecessary detail**

abstract classes in python:

python by default does not have abstract classes, but we can implement them by importing a certain module named abc, **a class is called abstract class if it has an atleast one abstract method**,and **abstract method is any method which will have a function declaration not function definition** (check the wheel\_count method in Vehicle class, any method which has declaration and no statements inside it is called abstract method).The abstract class is useful when there comes a scenario where the method from a parent class has to be definitely overridden.

from abc import ABC,abstractmethod #abc stands for abstract base classes.

class Vehicle(ABC):#NOTICE ABC sent as parameter

@abstractmethod #NOTICE the abstract method annotations here

def wheel\_count(self):

pass

def print\_nature\_of\_class(self):

print('the name of the class is Vehicle,nature of the class is to transport from one place to another')

class car(Vehicle):

def wheel\_count(self):

print('the car has 4 wheels')

car().wheel\_count()

car().print\_nature\_of\_class()

#if we don't declare wheel\_count method in car class then we will have the error-TypeError: Can't instantiate abstract class car with abstract methods wheel\_count

### How **abstraction is an endproduct of encapsulation**

We know that encapsulation is about simplifying behaviour and hiding data, by implementing one or both of these, we will define an object or a function, that **object or function is called abstraction**

### Why do we need abstraction

1. if you want to define a class with empty methods, which you want to inherit in its child class and fill the methods later
2. after creating an abstract class with few abstract methods and few non abstract methods, and if you want to override abstract methods in child class and you want to call non abstract methods of the parent class in the child class
   1. ex: look at the print\_nature\_of\_class above

### points to note in abstraction

* abstract method is a method which only has declaration and doesn't have definition.
* a class is called abstract class only if it has at least one abstract method.
* when you inherit a abstract class as a parent to the child class, the child class should define all the abstract method present in parent class.
* if it is not done then child class also becomes abstract class automatically.
* at last, python by default doesn't support abstract class and abstract method, so there is a package called ABC(abstract base classes) by which we can make a class or method abstract

### Difference between abstraction and encapsulation

**Encapsulation hides internal working so that you can change it later, hides details at implementation level.**

**Hides internal working of objects from outside world**

**Abstraction hides complexity by giving a more abstract(clearer) picture, abstraction hides details at design level**

**Focusses on relevant information by hiding unnecessary detail**

#### A real world example

Encapsulation – when you are driving a car, you pressed break #incomplete

###############################################

# annotations:

an annotation is nothing but additional information that is required to be explained for optional or compulsory data that is to be conveyed along with the process we perform

function annotations-

user defined data that adds some information to functions, adds information to function arguments and results, they are optional and makes no difference for interpreters

and makes no difference to interpreter.

ex1:

def add(a: int,b: int) ->int:

return a+b

add(1,2)#o/p:3

add('hi ','there!')#o/p:hi there!

ex2:

def add(a: list,b: (1,2,3)) ->lambda:-1:

return a+b

add(1,2)#o/p:3

above both the functions work perfecly well, it explains a is of int type, b is of int type and the return type is specified after the brackets.but as specified above, it does not mean that it should accept only int type, it will accept string type as well, and function annotations are for our refference to let that effect be on code we need 3rd party tools such as mypy for that

ex3:

def add(a: str,b: int) ->lambda:-1:

return a+b

message=f'''you are supposed to pass "a" variable of {add.\_\_annotations\_\_['a']}

and "b" variable of {add.\_\_annotations\_\_['b']}

and the return type is {add.\_\_annotations\_\_['return']}'''#here add.annotations is a dictionary which will store the data type of the parameter that is in function definition

print(message)

ex4:

def add(a: str,b: int) ->str:

'''you are supposed to pass "a" variable of string

and "b" variable of int

and the return type is string'''

return a+b

help(add)#this will return the function definition and the documented string it contains

ex5:

def add(a: str='no value passed,so default arugments are used here',b: int=1) ->str:

'''you are supposed to pass "a" variable of string

and "b" variable of int

and the return type is string'''

return a\*b

add()

###############################################

# snippets:

-print('hello',end=' 00p')

print('appended')#hello 00pappended

- the below code will take a list of strings as input and will print the count of unique strings and also count of each unique string.

from collections import Counter

p=[input() for \_ in range(int(input()))]

d = Counter(p)

print(len(d))

print(\*d.values())

-- checking wether a number is even or odd using bitwose manipulations

n=int(input())

if (n&1):

print(n,'is odd')

else:

print(n,'is even')

#this is a bit manipulation program and n if it is an odd number it will always have one at its right most position, and if n is an even number it will have 0 at its left most and and 0&1 will be 0

--check the difference between seconds of two dates:

pgm1:

from dateutil import parser#check what does this parser do

for \_ in range(int(input())):

d1 = parser.parse(input().strip())

d2 = parser.parse(input().strip())

print(abs(int((d2-d1).total\_seconds())))

pgm2:

from datetime import datetime as dt

fmt = '%a %d %b %Y %H:%M:%S %z'

for i in range(int(input())):

print(int(abs((dt.strptime(input(), fmt) -

dt.strptime(input(), fmt)).total\_seconds())))

input format-Sun 10 May 2015 13:54:36 -0700,Sun 10 May 2015 13:54:36 -0000

###############################################

# interview questions:

difference between list and tuple: tuple is faster than list,list is mutable,tuple is immutable(but if you try to place a list inside a tuple and then try muting it it will be mutable)

difference between dynamically-typed and statistically-typed:

--A language is statically-typed(if you don't have the need to declare the type of the variable while declaring it) if the type of a variable is known at compile-time instead of at run-time. Common examples of statically-typed languages include Java, C, C++, FORTRAN, Pascal and Scala

--Dynamically typed languages. A language is dynamically-typed if the type of a variable is checked during run-time, no need to expliciltly specify the type of data a variable holds. Common examples of dynamically-typed languages includes JavaScript, Objective-C, PHP, "Python", Ruby, Lisp, and Tcl

key features of python:

1:python is interpreted language, unlike c and its variants it code doesnt have the need to compile.

2:dynamically-typed

3:supports object oriented programming

4:many inbuilt functions

5:functions are first class functions,i.e,functions can be assigned to variables and can be returned from other functions or can be passed as a parameters to a functions,instance of an object type,can be stored in datatype such as lists.

6:writing python code is quick,but execution is slow when compared to complied languages such as c, but python offers inclusions of c based extensions so mostly no issues, one such example is nympy,its fast and most of the number crunching

7:is in many applications such as flask,django, pyramid

--what type of language is python?Programming or scripting?

python is a scripting language, it is a general purpose programming language(a language which is used for writing software in a wide variety of application domains)

--what are access modifiers or specifiers in python?

access specifiers help us organize and label data according to its scope and prevent unneccessary access

public access specifier:members of a class that are declared public are easily accessible from anywhere of the program , ALL DATA MEMBERS AND DATA FUNCTIONS ARE PUBLIC BY DEFAULT

protected access specifier: data members of a class that are only accessible to the classes that are derived from it are called protected members, a member is declared protected if it is preceded by an underscore('\_') (A PROTECTED PROPERTY IS ONLY INHERITED BY CHILDREN IN REAL LIFE)

private access specifiers: data members that are only accessible within the class and they are preceeded by a double underscore('\_\_')

what is an interpreted language , is python an interpreted language explain?

a language which is executed line by line WITHOUT GETTING COMPILED is called interpreted language, as it is not compiled the code should not be in MACHINE LEVEL before run time, python is interpreted language

--what is pep8?

pep stands for python enhancement proposal.it will have a set of conditions which will help python code for maximum readability.

--memory management in python is done through "python private heap space", all data structures and variables are stored in this heap space , programmer will not have access to this space and interpreter takes care of this, allocation of heap space to objects is done through memory manager and python has inbuilt garbage collector which will recycle unused memory so that it can be made available to heap

what is namespace?

it is a naming system used to make sure names are unique to avoid naming conflicts, it is an area where we create and store variables, it is of two types:

1:class namespace- used to store class variables or static variables

2:object/instance namespace- used to store instance variables

--what is pythonpath?

it is a environment variable, when a module is imported PYTHONPATH is used to check its presence in various directories,interpreter uses it to decide which module to load

--what are python modules?

python modules are files containing python code which is executable,the code can be either classess,functions or variables

ex:os,math,random,json,datetime.

--what are local variables,global variables?

variables which are declared outside a function are called global variables,they can be accessed anywhere in the program,local variables are declared inside the function and they are accessible inside the function,trying to access it outside the function will throw an error

--python is a case sensitive language, FOR EX; IF YOU USE PRINT IN CAPS INTERPRETER WILL NOT RECOGNISE.

--what is type conversion in python?

type conversion means converting data from one datatype to another.

ex:int(float('5.5')) will return 5 , other type convertrs are int,str,float,ord,hex,oct,list,tuple,dict,set,complex

--indentation is mandatory

--array and list hold data the same way,but array can only hold single data type,list can hold multiple data types

import array as arr

arr.array('i',[1,2,3,'p']) #here i stands for int and arr will accept only integers in list, but 'p' is string and it will raise an error

-- a function is a block of code executed when called

-- \_\_init\_\_ is a method or constructor called automatically each time to allocate memory so that a class is invoked to create an object or instance of it is created.

-- what is a lambda function?

it is an anonymous function which can have any number of parameters ,but has only one statement (an equation to use parameters and return a value)

syntax: function\_name=lambda <parameter1>,<parameter2>.... :one line code that is to be implemented in function

example: a=lambda x,y: x+y

print(a(3,2))#outputs 5

--what is self in python?

self variable in the init method refers to the newly created object, and in all other functions it refers to the object from which it is called.

-- can you shuffle a list?

from random import shuffle

x=[1,2,3,4,5]

shuffle(x)

print(x)

--difference between range and xrange

range xrange

supported in python 3 not supported

returns a range obj returns an xrange object

a list is created a generator is created

occupies more space occupies less space

slicing is possible slicing is not possible

--what are python iterators?

python objects which can be traversed through or iterated upon

-- what is yield in python?

yield is a keyword in python used to return a value from a function without destroying the state of the variables when the function is called the execution starts from last yield statemnt, if a function has a yield keyword then the function is called a generator.yield keyword in python is less known but has a greater utility one can think of, however it has its pros (it stores local variable states overhead of memory allocation is controlled,since old state is retained , flow doesnt start from the beginning and it saves time)and cons(if function calling is not called properly it will become erroneous, complexity is more and difficult to understand)

ex:# Python3 code to demonstrate yield keyword

# generator to print even numbers

def print\_even(test\_list) :

for i in test\_list:

if i % 2 == 0:

yield i

test\_list = [1, 4, 5, 6, 7]

print ("The original list is : " + str(test\_list))

print ("The even numbers in list are : ", end = " ")

for j in print\_even(test\_list):

print (j, end = " ")

-- what is pickling and unpickling in python?

pickle module accepts any python object and converts into string and dumps into a file using dump function, the process is pickling, the process of converting stored python object in string form to python object is called unpickling.

-- what are generators ?

functions which return iterable set of items are called generators

generators provided a space efficient method for data processing as only parts of a function are handled at a given point of time.

a generator function is defined like a normal function,whenever it needs to return a value it uses the yield keyword instead of return, when a function is using a yield keyword it is called generator.

generator object: a generator function returns a generator object, it can be called using "next" method(depricated now) of generator object or iterating the generator in a for loop.

def fib(limit):

a, b = 0, 1

# One by one yield next Fibonacci Number

while limit>0:

yield a

a, b = b, a + b

limit-=1

for i in fib(int(input())):

print(i)

--what is the way in which data is stored in lists and dictionaries?

in lists the data is stored in continous memory blocks and in dictionaries the data is stored in hash map functionality, what happens in background is performing hash map on a key will return the memory location of the value to which the key is linked to.

--what is dictionary?

dictionary defines one to one relation between one key and one or more values, key cannot be duplicate in dictionary.

-- what is difference between \*args and \*\*kwargs?

we use \*args when we dont know how many arguments are passed to a function, we use \*\*kwargs when we dont know how many keyword arguments are passed to a function.

-- len() in python?

used to determine length of a list or array or string.

-- whenever python exits all the memory is not deallocated because python has a few modules which have circular reference to other objects and objects that refer to global namespace are not always deallocated or freed, c library space will never be freed, due to its own efficient cleanup mechanism it will clear all possible unused memory allocations

-- types of ternary operators

if loop: a if a<b else b # if a is less then b(means the expression is true) will return a else b

tuple loop: (a,b) [a>b] # if a is greater than b (means the expression is true) will return b else return a

dictionary loop: {True:b,False:a} [a>b] # if the expression is true it will return True value and if false it will return false

nested if else: print ("Both a and b are equal" if a == b

else

"a is greater than b" if a > b else "b is greater than a")# if the cndition is true left part will be executed else the right part is executed

-- to remove a file:

import os

os.remove('filepath')

--Elements can be added to an array or list using the append(), extend() and the insert (i,x) functions

--Array or list elements can be removed using pop() or remove() and del method. pop returns the removed element remove funtion doesnot

snippets:

-- checking wether a number is even or odd using bitwose manipulations

n=int(input())

if (n&1):

print(n,'is odd')

else:

print(n,'is even')

#this is a bit manipulation program and n if it is an odd number it will always have one at its right most position, and if n is an even number it will have 0 at its left most and and 0&1 will be 0

--check the difference between seconds of two dates:

pgm1:

from dateutil import parser#check what does this parser do

for \_ in range(int(input())):

d1 = parser.parse(input().strip())

d2 = parser.parse(input().strip())

print(abs(int((d2-d1).total\_seconds())))

pgm2:

from datetime import datetime as dt

fmt = '%a %d %b %Y %H:%M:%S %z'

for i in range(int(input())):

print(int(abs((dt.strptime(input(), fmt) -

dt.strptime(input(), fmt)).total\_seconds())))

input format-Sun 10 May 2015 13:54:36 -0700,Sun 10 May 2015 13:54:36 -0000

# competitive\_programming\_tips:

--check the range function, it will not include end value i.e, list(range(5)) will give [0,1,2,3,4]it doesnt include 5 in it

--list(range(10)) will give [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],list(range(1,10+1)) will give [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(string.capwords('asdasf adda adcksj'))#o/p:Asdasf Adda Adcksj, main disadvantage here is it will swallow the extra spaces.

--import string

print(string.ascii\_uppercase) #o/p:ABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.ascii\_lowercase) o/p:abcdefghijklmnopqrstuvwxyz

print(string.ascii\_letters)o/p:abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.digits) o/p:0123456789

other operations on string module are string.hexdigits,string.octdigits,string.printable,string.punctuation,string.whitespace

--very very important in competitive programming

when you want first input into first list and so on...

l=list(range(11))

subli=3

l\_i=[[] for i in range(subli)]

count=0

for i in range(len(l)):

l\_i[i%subli].append(l[i])

l\_i

--very very important in programming

when you assign a list a to b as

a=[1,2,3]

b=a

b[0]='hello'

a,b

and if you print a, you will get ['hello',2,3] so it will be a major blunder(this happens because on normal assignment both lists will point to same memory and changes made using one variable will reflect in another as well)THIS NOT ONLY HAPPENS WITH LISTS IT HAPPENS WITH ALL MUTABLE VALUES, if you do assignment directly,so just use .copy() function which will avoid the mistake

a=[1,2,3]

b=a.copy()

b[0]='hello'

--l\_i=[[] for i in range(3)] #this method is prefered compared to the one below it as it refers each list to different memory.

l\_p=[[]]\*3

o/p for both:[[],[],[]]

--help(re.split) #will show all the methods related to it and explanation if exists

--num\_list=[10,20,30,40,50]

print("Map example")

print(list(map(lambda x:x\*2,num\_list)))#[10, 20, 30, 40, 50]

print("Filter Example")

print(list(filter(lambda x:x>30,num\_list)));#[40, 50]

print("Chain Example")

print(list(filter(lambda x:x>30,map(lambda x:x\*2,num\_list))));#[40, 60, 80, 100] in chain example execution happens from right to left here, first it gets multiplied by 2 and then it checks if it is greater than 30 or not

# Subprocess

This command is used to call external commands, if the external command fails, then no exception will be thrown by default, to raise an exception in case of error, you have to pass **check=true** as a parameter to subprocess.run command, if we assign the subprocess run command to a variable, then you can retrieve the error code for that by using <assigned variable>.returncode, to get the error you have to give <assigned variable>.stderr

Ex: p1=subprocess.run([‘ls’,’-lart’,’dne’], capture\_output=True, text=True)

p1.stderr

p1.returncode

Below are some commands that are run using linux commands

* subprocess.run(‘ls’)
* subprocess.run(‘ls -lart’, shell=True)
* p1=subprocess.run([‘ls’,’-lart’])
  + p1.args will return [‘ls’,’-lart’]
  + p1.returncode will return 0 i.e., means successfull
* p1=subprocess.run([‘ls’,’-lart’] , capture\_output = True)
  + capture\_output=True will prevent from sending the stdout and stderr to console and stores it in p1
  + print(p1.stdout) will store it as a string with escape sequences, p1.stdout.decode() will print it in normal format
* p1=subprocess.run([‘ls’,’-lart’], capture\_output=True, text=True)
  + This will remove over heads shown in above and will store printable string directly to print(p1.stdout)
* P1=subprocess.run([‘ls’,’-lart’],stdout=subprocess.PIPE, text=True, stderr=subprocess.PIPE)
  + The above command will have either the stdout or stderr to be stored in pipe. i.e., if the command successfully executes the value will be stored in stdout and can be retrieved using p1.stdout and if the passed command is having error, then the output will be stored in p1.stderr
* We can write the standard output, standard error in to a file
  + with open ('output.txt','w') as f:

subprocess.run(['ls','la'],stdout=f,stderr=f,text=True)

* if you want to simply redirect the error, then you can use stderr=subprocess.DEVNULL
  + p1=subprocess.run([‘ls’,’la’],stderr=subprocess.DEVNULL)
* passing one output of a subprocess to another subprocess as input
  + here in this example we are extracting content of a output.txt and then searching for word line in the second subprocess

ex1:

p1=subprocess.run(['cat','output.txt'],capture\_output=True,text=True)

p2=subprocess.run(['grep','-n','lines'], capture\_output=True, text=True, input=p1.stdout)

p2.stdout

ex2:

p1=subprocess.run('cat output.txt|grep -n line', capture\_output=True, shell=True, text=True)

p1.stdout

**ex1 and ex2 both are of same approach**

* A

# Data Flair

--Functions:A function in Python is a collection of statements grouped under a name.

--Classes:Python is an object-oriented language. It supports classes and objects. A class is an abstract data type(Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of value and a set of operations).In other words,it is a blueprint for an object of a certain kind. It holds no values. An object is a real-world entity and an instance of a class.

ex:explaining about a parrot is called class, a real parrot is an object

--Modules:A Python module is a file containing python statements and definitions

--Packages-Python package is a collection of related modules (note the difference between package and module even in alhabets first M(Module) then P(package) i.e, collection of modules is called package)

--in python doc string(documentation string) is a string between three double quotes/single quotes (ex:'''this is a doc string''') used for documenting information about a module i.e, what does this module do,it can be retrieved using syntax: module/function.\_\_doc\_\_ ,difference between docstring and comment is that we cannot retrieve whatever is present in comment by using python commands but we can retrieve the information in docstring using the above mentioned command, the data stored is treated as a string datatype.

print(method/library.\_\_doc\_\_) will print the documentation details of that method/library,documentation info is declared in triple quotes right after the function is declared(starting of the loop)

ex:list.\_\_doc\_\_

--python features are:interpreted,open source,easy to learn,vast packages,GUI programming can be done,Object oriented,portable,supports multi programming paradigms,python is "extensible"(you can write some of your code in other languages while coding in python) and "embeddable"(code of python can be used in other language programs)

-multiprogramming paradigms supported by python:it supports

procedural(this programming paradigm divides the program in to routines or functions, simply put it tells computer step by step of solving a program or solves the program using procedures or functions),functions inside the class in object oriented programming and are called methods)

object oriented:

the process of solving a programming problem by creating objects is called object oriented programming.,

functional (a paradigm in which we try to bind everything using pure mathematical functions,typically has usage of iterators and generators,

ex:L = [1, 2, 3]#or you can use "reduce" example and

it = iter(L)

it

<...iterator object at ...>

it.\_\_next\_\_() # same as next(it)

1

next(it)

2

next(it)

3

next(it)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

StopIteration

imperative programming(this programming paradigm uses statements that can change the state of a program) paradigm:

procedural and object oriented programming approach come under imperative programming approach, it mainly focusses on how to complete a program.this uses block approach(group of statements under a name-> objects or methods)

ex: sum = 0

for x in my\_list:

sum += x

print(sum)#even if the length varies the sum will be calculated this is an example for imperative programming.

-Python Django is a free and open-source framework written in Python and is the most common framework for Python. It allows you to create database-driven websites. It follows the DRY Principle (Don’t Repeat Yourself). This is a design philosophy that keeps code simple and eloquent.

Like Django, Flask is a web framework written in Python itself. It is a micro framework because it does not need certain libraries and tools.

django,pyramid and flask are python frameworks( a framework is a combination of certain components and packages,a framework will help programmers to focus on high level functionality as the framework takes care of the low level functionality by itself, a framework will let us save time from doing the routine stuff as it already has those things configured, in other words a framework is a software providing generic functionality which can be changed using few lines of code thereby developing application specific framework) which help in developing webapps

-Data science:earlier python was used for automation,building applications and writing scripts,but below packages changed python to favourites of data science enthusiasts

pandas:Data analysis

numpy(implemented in C, so it is quicker):scientific and numerical calculations

matplotlib,seaborne:for data visuvalizations

scikitlearn:ML

tensorflow(an array of n dimensions),keras:ML and DL

-for artificial intelligence:

NLTK:Natural Language ToolKit

OpenCV:open source computer vision library

microsoft cognitive toolkit: deep learning framework

--Automation-python can be used to automate test cases,scrape web and also automate mundane tasks,it also can read all types of files easily.

(frameworks and libraries used for this are -robotFramework,Pytest,Pyunit,Selenium Python)

-python has a lot of micro frameworks , mega frameworks and web frameworks.

-CPython:This is the most widely accepted implementation of Python. It is written in the language C, and is an interpreter.

-PyPy is Python implemented in Python.

--positional arguments example:complex(3,6)->3+6j,complex(6,3) will have 6+3j as value in the sense-the parameter which is first passed is taken as real number and the latter one is an imaginary number, (1+2j).imag will return 2 and (1+3j).real will return 1

phase and modulus of complex number z=3+2j (or any complex number can be calculated using the code below)

from cmath import phase#cmath module will have mathematical functions related to complex numbers

if \_\_name\_\_=='\_\_main\_\_':

input\_complex=complex(input());

print(abs(input\_complex))

print(phase(input\_complex))

keyword arguments:if you want 3+6j as output then example(complex(imag=6,real=3)) will give 3+6j as output ,irrespective of position it will take the value as input

def dunc(a,b,/,c,d,\*,e,f)#(python 3.8 feature)the parameters before / should be definetly positional and the parameters after \* should definetly be keyword arguments

valid examples:

dunc(1,2,3,4,e=5,f=6)

dunc(1,2,c=3,d=4,e=5,f=6)

invalid examples:

dunc(a=1,b=2,3,4,e=5,f=6)

dunc(1,2,c=3,d=4,5,6)

--pickle :

Python pickle module is used for serializing and de-serializing a Python object structure, an object is pickled in python so that the object which is initially stored in python memory to a file so that it can be accessed from a different source or progrmamming langauge, pickling is nothing but serializing the object and then writing it in to a file, to explain in detail, picking is the process of converting an object in to a character stream, so further in future, we can use this character stream to reconstruct the object, marshal module also serves more or less the same purpose.

example program:

import pickle

def storeData():

# initializing data to be stored in db

Omkar = {'key' : 'Omkar', 'name' : 'Omkar Pathak','age' : 21, 'pay' : 40000}

Jagdish = {'key' : 'Jagdish', 'name' : 'Jagdish Pathak','age' : 50, 'pay' : 50000}

# database

db = {}

db['Omkar'] = Omkar

db['Jagdish'] = Jagdish

dbfile = open('examplePickle.txt', 'ab')# Its important to use binary mode

# source, destination

pickle.dump(db, dbfile) #deserializing, it is not mandatory to deserialize it to a file, we can just assign it to a variable vari=pickle.dump(db)

dbfile.close()

def loadData():

dbfile = open('examplePickle.txt', 'rb') # for reading also binary mode is important

db = pickle.load(dbfile)#serializing

for keys in db:

print(keys, '=>', db[keys])

dbfile.close()

if \_\_name\_\_ == '\_\_main\_\_':

storeData()

loadData()

--fstrings in python support debugging,python 3.8 allows fstrings with "=" operator

example:

A=5

print(f"{A=}")

--python 3.8 supports final keyword:

a class declared as final will not be allowed to get inherited

a variable declared final will not be allowed to get reassigned

a method declared as final will not be allowed to get overridden

note: a method and class will use annotations as @final,a variable will use Final

-@final Decorator (classes, methods)

from typing import final

@final

class Base:

# Cannot inherit from Base

class Base:

@final

def foo(self):

# Cannot override foo in subclass

-Final annotation

from typing import Final

PI: Final[float] = 3.14159 # Cannot set PI to another value

KM\_IN\_MILES: Final = 0.621371 # Type annotation is optional

class Foo:

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

self.bar: Final = "baz" # Final instance attributes only allowed in \_\_init\_\_

--p=0

print(p==False)# will return true as it forcechecks the type

--syntax error:EOL while scanning the literal->indicates that interpreter is unable to find the end of the line

--python doesn't mandate semicolons,each line is a statement in python.to get some work done in multiple lines we use:

ex1:print("Hi\

how are you?")

ex2:a\

=\

10

will simply assign 10 to a

ex3:

a multiline string or docstring can use multiple lines without any backward slash

ex:print("""hi

how

are

you

""")

--we can use multiple statements in a single line using semicolons

a=1;b=2;c=3

print(a,b,c)

--run this(these below print statements will make us include apostrophe or inverted quotes as part of a string while printing)

print('s"hi"')#output:s"hi"

print("s'hi'")#output:s'hi'

print("s'hi")#output:s'hi

print('s"hi')#output:s"hi

--fstrings in python will take the print value of the variable in the print satement but we mention the name of identifier/variable in print statement

ex:x\_value=10

print(f'the value of x is {x\_value})

# Class

getattr(obj or self,variable)#will give the value of the variable in the passed object.

setattr(obj or self,variable,value)#will set the passed value to the variable in the given object.

ex:setaattr(obj/self,'%s\_pv'%mode,value)#i think mode will be replaced by %s

-- functions are mini programs in a program

# psychology of python

in python there will be numerous functions coming each and every year and numerous packages coming every few weeks/months, so based on the requirement we have to search and choose the right package and then search for appropriate method in the package by using methods such as dir(package or method) choosing the right package/method and then using shift+tab in jupyter or doc string of the method to pass appropriate parameters.

# Numpy:

* A linear algebra library for python, almost all libraries of pydata ecosystem consider numpy as its building block, it is incredibly fast and has bindings to c libraries.
* Numpy arrays come in 2 flavours- vectors (1d- arrays) and matrices
* np.ones, np.zeros, np.random.randn will accept tuple of integers as input for a 2d array, but np.random.rand will accept only sequence of integers separate by a comma for min and max, then size as tuple. np.random.randn can be passed to dataframe for generating a random instant dataframe
* ex: np.ones((3,2), np.random.rand(min int(inclusive), max int (exclusive),(row x columns))
* to get the row count or column count, you can use .shape method for numpy array or pandas dataframe
* most commonly used array operations in numpy- reshape, min, max, argmin, arfmax, shape, dtype
* numpy is famous for its broadcasting

* + 
* Slicing in numpy arrays is similar to that of lists
* But a copy of array you get by slicing is not a separate array, but just a view of it, **therefore if you make any changes to the new sub array you get by slicing, then the parent array from which you sliced will also have those changes reflected.**
  + **To avoid that you can use <np array>.copy method**

* + 
* To access an element at ith row and jth column of a 2d array, then you can give arr[i,j]
* You can also retrieve the part of the matrix

* + 
* Using comparison operator on a numpy array in broadcasting manner!

* + 
* Operations on numpy array: you can perform element wise operation on a numpy array, i.e.,
  + if you perform addition on 2 numpy arrays, then element at 0th index of arr1 is added with element at 0th index of array 2
    - Addition, division, multiplication, subtraction are supported
  + Broadcasting is also supported at array level
    - +,\*,/,-,\*\* are supported
* When you try to divide zero by anything or anything by zero, numpy array operation will not result in an error, simply it just gives warning
* A universal function (or ufunc for short) is a function that operates on ndarrays in an element-by-element fashion, supporting array broadcasting, type casting, and several other standard features.
  + Go through the link https://numpy.org/doc/stable/reference/ufuncs.html
  + List of math operations supported by universal functions: <https://numpy.org/doc/stable/reference/ufuncs.html#math-operations>
* np.random.seed will let you generate the same random numbers

## numpy misc to be arranged above

--np.insert(features, 0, input\_data[label\_column], axis=1) #have to play with it and check

--computation with numpy array is 200x faster than lists

--a nD numpy array is alled tensor

--import numpy as np

b=np.arange(0,10,1)#(start,end,step)#similar to for loop end is not included in array

b->o/p:array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

--a=np.linspace(0,1,6)#1 at ending is fixed total 6 positions remaining 5 positions (0+1)/5=0.2 so interval is 0.2 starting with 0

a#o/p:array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])

--a=np.eye(3)#o/p:[[1., 0., 0.],[0., 1., 0.],[0., 0., 1.]]

b=np.eye(3,2)#o/p:[[1., 0.],[0., 1.],[0., 0.]]

c=np.diag([5,6,7,8])#o/p:[[5, 0, 0, 0],[0, 6, 0, 0], [0, 0, 7, 0],[0, 0, 0, 8]] notice the square braces

d=np.diag(c)#o/p:[5, 6, 7, 8]

a=np.random.randn(3,3)#gives random values of shape 3x3

a#o/p:[[ 0.26309984, -0.68504932, -0.05957326],[-0.25660805, 0.31861719, 0.65046857],[-2.41144948, -0.59969052, 1.15969629]]

a.dtype#will give datatype that a given numpy array holds

a=np.ones((3,3))#o/p:[[1., 1., 1.],[1., 1., 1.],[1., 1., 1.]],notice double braces

b=np.zeros((3,3))#o/p:[[0., 0., 0.],[0., 0., 0.],[0., 0., 0.]]

c=np.arange(10,dtype='int');print(c)#o/p:[0 1 2 3 4 5 6 7 8 9]

a=np.diag([1,2,3])

a#o/p:[[1, 0, 0],[0, 2, 0],[0, 0, 3]]

a[2,2]=5

a#o/p:[[1, 0, 0],[0, 2, 0],[0, 0, 5]]

--a=np.arange(1)

b=a[1:8:2]

c=a[5:]

print(a,b,c)#o/p:[0] [] []

or

a=np.arange(10)

b=a[::2]

c=a[::]

print(a,b,c)#o/p:[0 1 2 3 4 5 6 7 8 9] [0 2 4 6 8] [0 1 2 3 4 5 6 7 8 9]

--a=np.arange(0,10,1)

b=np.arange(0,10,2)

np.shares\_memory(a,b)#o/p:False

a=np.arange(0,10,1)

b=a[0::2]#this is a view,it shares common memory

c=a[0::2].copy()#this is a copy,it doesn't share common memory

np.shares\_memory(a,b)#o/p:True

np.shares\_memory(a,c)#o/p:False

--a=np.random.randint(0,20,10)#(start,end,count)will have elements from start to end-1 and the number of elements is count parameter

a#output:[ 8, 11, 18, 1, 18, 11, 1, 15, 0, 16]

--a=np.random.randint(0,20,10)

mask=(a%2==0)

a\_b=a[mask]#mask creates copy not views

a\_b

--a=np.array([0,10,20,30,40,50,60,70,80,90])

b=a[[0,3,2,4,7]]#an np array with 2 square brackets will deal with indices of nparray

a[[9,7]]=-200

a,b#array([ 0, 10, 20, 30, 40, 50, 60, -200, 80, -200]),array([ 0, 30, 20, 40, 70])

b=a+1#arithmetic operators operate elementwise on numpy array

c=a-b

a,b,c#array([ 0, 10, 20, 30, 40, 50, 60, -200, 80, -200]),array([ 1, 11, 21, 31, 41, 51, 61, -199, 81, -199]),array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1])

--a=np.array([147, 176, 105, 35, 229, 90, 140, 272, 44, 295])

b=a[[9,3,-4,5,1,6,7,-1,2]]#values in double square brackets indicate indices of the array a,np array b will hold corresponding indice vales declared in array a

b

--a=np.array(any\_list)#will create an array

--a=np.array([147, 176, 105, 35, 229, 90, 140, 272, 44, 295])

b=a+1#o/p:[148 177 106 36 230 91 141 273 45 296]

c=a-1#o/p:[146 175 104 34 228 89 139 271 43 294]

#all arithmetic operators operate element wise(corresponding position wise)

print(b,c)

--a=np.diag([3,5,7])

a[[1,2]]=4#will make second and third rows values 4 as index starts from 0

--a=np.diag([3,5,7])

a[1,2]=4#will make secondrow and third column value 4 as index starts from 0

c=a\*a# it can also be declared as a.dot(a)

c#o/p:[[ 9, 0, 0],[ 0, 25, 16],[ 0, 0, 49]]

--a=np.array([1,2,3,4])

b=np.array([5,2,2,4])

a=np.array([1,2,3,4])

print(a>b) #output:[False False True False]

print(a==b)#output:[False True False True]

np.array\_equal(a,b)#will return true or false

np.array\_equal(a,c)#will return true or false

np.logical\_or(a,b)#logical op

np.logical\_and(a,b)

np.sin(a)

np.log(a)

np.exp(a)#all above operations will return np array as output

--shape mismatch error will come when we operate on 2 uncompatible arrays

--flatten?

--ravel?

# Pandas

* An open source library built on top of Numpy
* Allows fast analysis, data cleaning, data preparation
* Can work with data from wide variety of sources
* Allows visualization of data

## Series

* Similar to numpy array and is built on top of numpy array
* Holds almost all types of objects as values, ex: dict ,list, numpy array
* Numpy array does not support indexes, series supports indexes and also allows to label those indexes
* Default index starts numbering from 0, check labels a, b, c from below image
* Will have only one column
* We can apply aggregate functions such as sum, max and others
  + Text

    Description automatically generated
* Integers will be converted in to float in order to not to lose information
* You can also add values of 2 different series with matching labels, if any of the labels is missing, the resultant series will have nan as the value for those lables.

## Dataframes:

* Dataframes are **flexible and easy to use**, the reason is **when you operate on a dataframe and most of the times the result is still a dataframe and on that dataframe you can still perform a lot of other functions**
  + Example: Groupby, transpose and max are used on a dataframe but we an use all of them at once
    - Ex: df.groupby(‘Country’).max().transpose()[‘fb’]
    - This syntax will group all countries in to one row and include all columns it can i.e., if you are using a max function it will include employee name who’s salary is maximum. If you are using a sum function, then it cant return employee name column
* Below 3 declarations are same
  + df=pd.DataFrame(data=np.random.randn(5,4),index=['A','B','C','D','E'],columns=[‘W’,’X’,’Y,’Z’])
  + df1=pd.DataFrame(np.random.randn(5,4),['A','B','C','D','E'], [‘W’,’X’,’Y,’Z’]))
  + df2=pd.DataFrame(index=['A','B','C','D','E'], columns=[‘W’,’X’,’Y,’Z’]),data=np.random.randn(5,4))
* df.head() will give the first 5 rows, df.tail() will give the last 5 rows
* df.info() will give basic information such as row count, datatype and other information such as memory usage, count of each datatype columns, ex: 3integer type cols, 2 char columns
* columns and rows are treated as pandas series when tried to extract from dataframe
* axis = 0 means it refers to index (rows) axis=1 means it refers to data (columns), the explanation behind that will relate to df.shape syntax which gives a tuple with rows at 0th index and columns at 1st index ex: (4,5)
  + when I refer to some method that can be applied across rows or columns, then it means you can alter their implementation by passing axis = 0 or 1
* each column or row in dataframe is a series
* df[‘W’] is same as df.W but the first declaration is preferred as the second column might lead to overlap of column name with df methods
  + ex: if the column name is abs then df.abs is the way to retrieve the column and there is already an inbuilt method with it
* df[‘new\_col’] = df[‘existing\_col1] + df[‘existing\_col2’] is valid and is used to generate new columns
* df.drop(<column or row name in quotes>, axis=<0 for index, 1 for data>, inplace = <True if you want to have changes reflect in df, else if you want to assign it to new df and you want old df to be exactly same, then use false>)
* to access a one column you can use df[‘column1’] (this returns you a pandas series) or df[[‘column1’]] (this returns you a pandas dataframe), to access more data column, you can actually use df[[‘column1’, ‘column2’]] # **returns df pandas obj, therefore we can apply all methods on this obj which we apply in general to a df**
* two ways to access rows
  + df.loc: you have to give the row name ex: student id
    - you pass row index values, it is nothing but the values in the index of the dataframe, you can also slice dataframe based on index values
      * if the index is having date values, you can slice the dataframe as df.loc['2015-01-01':'2016-12-31'].std() both start and end values passed in slicing are inclusive
    - df.loc[‘row1’] -> returns df series obj
    - df.loc[[‘row1’]] -> **returns df pandas obj, therefore we can apply all methods on this obj which we apply in general to a df**
    - df.loc[[‘row1’,’row2’]] -> returns df pandas obj
    - sal.loc[sal['EmployeeName']=='JOSEPH DRISCOLL']['JobTitle'] #this will let sal['EmployeeName']=='JOSEPH DRISCOLL'] return a Boolean series which will be feeded as input to sal.loc which will return a df with entries whose name has JOSEPH DRISCOLL, and job title of the joseph is retrieved
  + df.iloc: I in iloc stands for index,you have to give numerical index ex: 0,1,2
    - df.iloc[‘1’] -> returns df series obj
    - df.iloc[[‘1’]] -> returns df pandas obj
    - df.iloc[[‘1’,’2’]] -> returns df pandas obj
* **to access a particular value at ith  row and jth column use df.loc[‘I’,’j’]**
* **to access a particular subset of rows and columns use df.loc[[<list of rows to include>],[<list of columns to include>]]**

### CONDITIONAL SELECTION IN DATAFRAMES

* + if you want to only have positive values in a df and negative values as NAN, you can simply give df[df>0], if you want to remove negative values based on a particular column, then you can give df[df[‘w’]>0], if you want to return only a particular columns from dataframe after removing negative values from another df, then use df[df[‘w’]>0][[‘w’,’x’]]
  + **to include multiple conditions** df[(df[‘W’]>0) and (df[‘Z’]>2)] will give an error because normal **and** cant operate on series, so we have to use **& for and** operation **| for or** ex: df[(df[‘A’]<5) | (df[‘C’]>1)]

### index

* df.reset\_index this method will convert the existing index as a column of the existing df, use inplace=True if you want to reflect the changes in underlying dataframe
  + df.reset\_index(inplace=True)
* df.set\_index(existing\_column, inplace=True/False) this method will take existing column as input and that existing column is made index.
* df.index.names is used to get names of the indexes in df
* we can create multi indexes in a dataframe, for that you need to create a MultiIndex (check image below) dataframe object and pass it to dataframe creation syntax, to access sub index of the below DF, you need to use df.loc[‘G1’].loc[1]
  + Graphical user interface, text, application

    Description automatically generated
* In the below image, then you can use df.xs method
  + Graphical user interface, application

    Description automatically generated
  + To access a group g1 as a whole, you can use df.loc[‘G1’] or df.xs[‘G1]
  + if you want to access rows whose num index has value 1, you can use df.xs([‘G1’,1]) or df.xs(1,level=’Num’)
  + df.xs can not be used to set values

#### multi indexing at row and column level

easy way to create multi indexing at row level and column level at the same time by creating arrays for index and column names

row\_arrays = [

np.array(["row1", "row1", "row2", "row2"]),

np.array(["one", "two", "one", "two"]),

]

col\_arrays = [

np.array(["col1", "col1", "col2", "col2", "col3", "col3", "col4", "col4"]),

np.array(["one", "two", "one", "two", "one", "two", "one", "two"]),

]

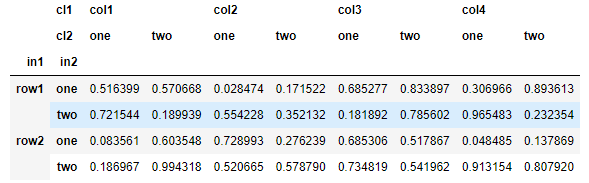
df=pd.DataFrame(np.random.rand(len(row\_arrays[0]),len(col\_arrays[0])),index=row\_arrays, columns=col\_arrays)

df.index.names=['in1','in2']

df.columns.names = ['cl1','cl2']

df.head()

output:



in the above code, we have declared a list of numpy arrays for row and columns as **row\_arrays** and **col\_arrays**, we passed them as **index** and **columns** to dataframe, mind that whenever you are using multi level indexing at row level or column level, together or each alone their corresponding length should be matching, i.e., in **row\_arrays** list you have multiple np arrays, length of each np array should match the **count of rows** in the corresponding dataframe

##### setting existing multiple columns as indexes

setting existing multiple columns as indexes will make accessing values for a particular set of column attribute related elements, for example let’s say your data is like below

d = {'num\_legs': [4, 4, 2, 2],

'num\_wings': [0, 0, 2, 2],

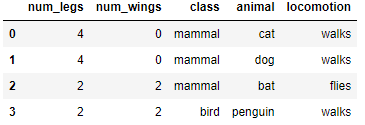
'class': ['mammal', 'mammal', 'mammal', 'bird'],

'animal': ['cat', 'dog', 'bat', 'penguin'],

'locomotion': ['walks', 'walks', 'flies', 'walks']}

df = pd.DataFrame(data=d)

df



If you want to fetch all occurrences for a mammal which walks or a num\_legs for a cat which walks, then we need to set those multiple columns as indexes by using set\_indexes method which needs to be reassign it to the same df or pass inplace = true as parameter

d = {'num\_legs': [4, 4, 2, 2],

'num\_wings': [0, 0, 2, 2],

'class': ['mammal', 'mammal', 'mammal', 'bird'],

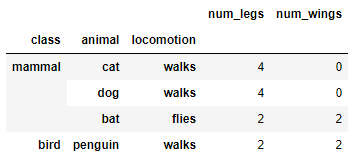
'animal': ['cat', 'dog', 'bat', 'penguin'],

'locomotion': ['walks', 'walks', 'flies', 'walks']}

df = pd.DataFrame(data=d)

df.set\_index(['class', 'animal', 'locomotion'],inplace=True)

df

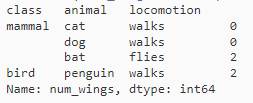


#now if we want the num\_legs of a cat which walks

df.xs(('cat','walks'),level=[1,2])['num\_legs'] #because cat is an animal which is at index 1 and walks is a values of locomotion which is at index 2

Get values at specified column and axis

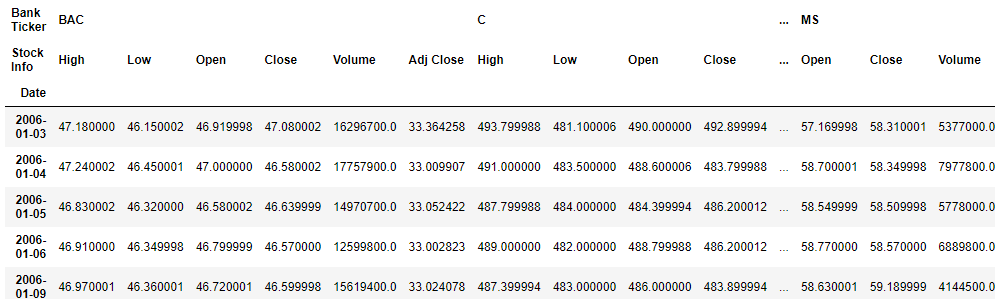
**>>>** df.xs('num\_wings', axis=1)



###### Note

When you don’t give any axis it will by default act on rows, if you want to act on columns, then you need to pass axis = 1 parameter to df.xs() method

bank\_stocks.head()



bank\_stocks.xs(key='Close',axis = 1,level="Stock Info").max()

bank\_stocks.xs(('Close'),level = 1, axis = 1).max()

you can see 2 levels for columns above level 0: bank ticker, level 1: Stock info, so you can either pass level value (0 or 1) or name of level

both the above syntaxes are same

### missing values

* df.dropna:
  + axis=0(for dropping rows), axis=1 for dropping columns
  + thresh=<int>, will drop rows, if the count of na’s is greater than or equal to specified int
* df.fillna(value=<any value that you want to fill in place of np.nan>
  + to fill a specified column, df[‘A’].fillna(value=df[‘A’].mean()

### group by

group by will group similar columns based on the unique values in the column provided and act on remaining numerical columns to give output

ex: grouping people based on surname and getting average income of the family

* + Groupby, transpose and max are used on a dataframe but we an use all of them at once
    - Ex: df.groupby(‘Country’).max().transpose()[‘fb’]
    - This syntax will group all countries in to one row and include all columns it can i.e., if you are using a max function, it will include employee name who’s salary is maximum. If you are using a sum function, then it can’t return employee name column

#### Generating a matrix plot (viable for heat map) using groupby method and unstack

df.groupby(by=['Hour','day']).count()['twp'].unstack(level=-1)

this generates a matrix plot when you use unstack

##### level = -1

will make the first value of list assigned to **by** keyword above i.e., **hour (unique values)** as index and **day** **(unique values)** as column labels

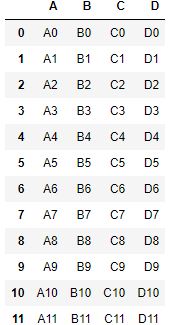
##### level=0

will make the first value of list assigned to **by** keyword above i.e., **hour (unique values)** as column labels and **day** **(unique values)** as index

### Merging, joining and concatenating

#### concatenation

Concatenation in dataframes is much similar to concatenation of strings except that here we can concat one dataframe to another by attaching it as

* Horizontally, i.e., as extra rows by giving axis=0, this is **default** and for this to happen, you need to match dimensions of both dataframes
  + 
* Vertically, i.e., as Extra columns by giving axis=1
  + Table

    Description automatically generated

while concatenating, in the place of missing values, it will fill with nan

pd.concat(< list of df’s> ,axis)

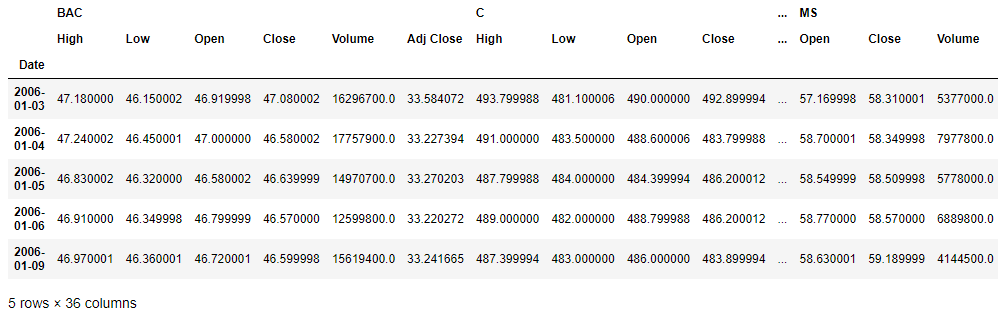
##### note

when you are concatenating one group of columns with another group of columns and you want to have different common value for each group of column, then you can pass keys

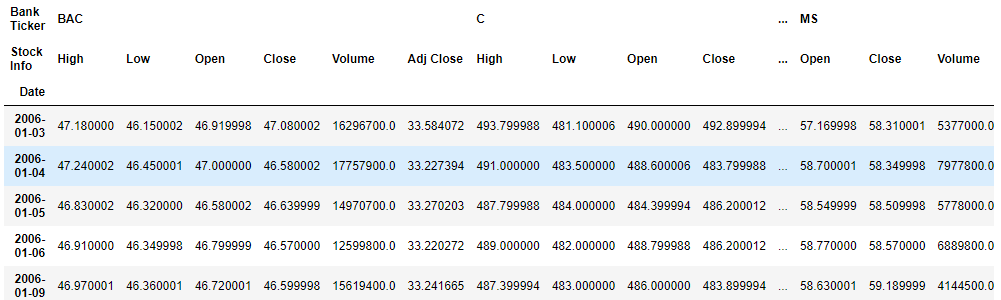
bank\_stocks=pd.concat(<list of dataframes you want to merge>,axis=1,keys=<list of different names you want to give to the dataframes passed in first arguments>)

here we gave axis=1 which means we are joining across columns, if you give axis = 0, you will join across rows

now you will be having 2 levels of variables as column names which will look like this



If you want to give names to them, you can use bank\_stocks.columns.names = ['Bank Ticker','Stock Info'] # as we have 2 column levels



#### merge

as the name suggests, merging means merging 2 dataframes in to one similar to the logic of merging 2 tables in SQL

pd.merge(<dataframe 1>, <dataframe 2> , how= <inner (default), outer, left, right, cross>, key =<a column or a list of columns>)

inner merge will only return matching columns, whereas the other merging methods such as left, right , outer will fill nan for cells when it doesn’t have any data related to them

#### join

a convenient method to attach columns of one dataframe to another, even with the mismatching index to result in a single dataframe

same as merge, but instead of key we use index here

ex: df1.join(df2, how= <inner (default), outer, left, right, cross>, key =<a column or a list of columns>)

#### difference between join and merge

both are similar to join implementation in SQL. But,in merge you specify the column based on which you want to perform the merge, in join merge is performed based on both the indexes of the data frame

#### operations

##### unique

this will return unique values for a given column

1. df[‘col’].unique()
2. if the col name is age you can also give df.age.unique()

##### nunique

this will return the count of unique values for a given column

df[‘col’].nunique()

##### value\_counts

this is used when you want to return the count of each unique value of a given column, lists things automatically in descending order

df[‘col’].value\_counts()

##### applied methods

for a given column, you can apply a function **which can be custom like multiply each element by 2, or an inbuilt function like lambda or len**, which will be broadcasted to all cells in the column

ex: def times2(num):

return num\*2

df['col1'].apply(times2)

* df[df[‘expiry date’].apply(lambda x: x.endswith(’25)].shape[0] # this will fetch all the rows of the df whose expiry date is of year 25 and it will return the count of rows
* df[‘mail’].apply(lambda x: x.split(‘@’)[1]).value\_counts().head() # **note that this is different from above and this will just return a pandas series and then performing value\_counts and head is a known part, a simple, beautiful yet complex example 😊**

##### df.drop

used to delete index or column and inplace is applicable

##### df.index, df.columns

will return index/ column names of a df

##### df.sort\_values

sort by values along either axis, even if the columns rejig the index values stay attached to rows

df.sort\_values(by=’col or row name’, axis=<0 or 1>)

##### df.isnull

will return a Boolean df having true for cells holding nan value

##### sal['BasePay'].mean()

will return the mean or average of basepay

##### sal[‘BasePay’].max()

will return the maximum of BasePay

##### df.pivot\_table

if we want to check the relation between unique values of one column with unique values of another column and declaring others as indexes (if needed)

import pandas as pd

data = {'A':['foo','foo','foo','bar','bar','bar'],

'B':['one','one','two','two','one','one'],

'C':['x','z','y','y','x','y'],

'D':[1,10,2,5,4,1]}

df = pd.DataFrame(data)

df.pivot\_table(values='D',index=['A', 'B'],columns=['C'])

note: columns A and B are combinedly made index and unique values of column C are made columns (x,y,z) in this case, values of d are populated as values, if there are more than one value possible for an item, then average of the values is considered.

##### df.map

dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}

df['day']=df['Day of Week'].map(lambda x:dmap[x])

# the above function will

##### df.corr()

this method will check correlation of each (afaik numerical) column with all columns (afaik numerical), example: if there are 3 columns correlation of a,a a,b a,c b,a b,b b,c c,a c,b c,c will be checked will be represented in a 3x3 matrix

below example gives the correlation between Hour and month of a df

Graphical user interface, text

Description automatically generated with medium confidence

##### df.std()

will return standard deviation for each column in the dataframe

or you can get std dev for a particular column as well

ex: returns['BAC return'].std()

##### Group by

##### df.transpose

transpose index and columns of the dataframe

##### input and output

pandas reads and writes data in to different formats, majorly used are as below

* Excel
* Csv
* Html
* Sql

In excel pandas thinks of excel as a workbook as a group of sheets and each sheet is treated as a dataframe

###### pd.read\_html

this will take URL as input and will return a list of tables as dataframe objects to the output

###### pd.read\_sql

pandas is not well suited to read from SQL as there are wide variety of SQL engines such ah mySQL, postgres, sqlite

we need to search for drivers as there are different drivers for different engines

###### pd.read\_csv

this will read the excel file and convert it as dataframe

ex: pd.read\_csv(<path>, index\_col=0) #index\_col = 0 will make the first col as index

## pandas built-in data visualization

some methods used for visualization in matplotlib can directly be called using pandas objects, these visualization methods use matplotlib package under the hood, so therefore you can pass parameters which you use in **matplotlib** and **seaborn** methods

prefer this method of pandas implementation only when you want quick analysis, else if you want to customize plots or you can spare some time for these analysis, prefer seaborn or matplotlib

as seaborn has better visualizations and by default pandas uses matplotlib under the hood, using **import seaborn as sns**, will make the visualizations look much better because sns is better and built on top of matplotlib

you can use a df or column of a df to do these visualizations based on need and supported type

df[<col>].hist(bins=20)

df[‘A’].plot.kind(‘hist’,bins=20) is same as df1['A'].plot.hist(bins=20) and the second declaration is most useful

few examples:

df2.plot.area(alpha=0.1)

df2.plot.bar(stacked=True) #if there are multiple columns and multiple indexes, then you will have indexes on x axis and multiple bar plots(because of multiple column values) for each index value ,stacked=True will let one bar stacked on another for the same index

df1.plot.line(y='B',figsize=(12,6),lw=1) #notice you are passing matplotlib arguments such as figsize and lw

df1.plot.scatter(x='A',y='B',s=df1['C']\*100)

## df snippets

* To make the first row as header

# new\_header = X\_train.iloc[0] #grab the first row for the header

# X\_train = X\_train[1:] #take the data less the header row

# X\_train.columns = new\_header #set the header row as the df header

## pandas misc

* df.pop a easy method which is used to remove a column from a df
* if you are going to read a file using pd.read\_csv method and in the file you don’t have column names, but you know the column names and if you already stored them in list, you can pass that list as a parameter to the read\_csv method like below
  + df = pd.read\_csv(‘df\_without\_column\_names.csv’, names = <list of names that you want to assign as columns to the df being used> , header = None)
* Difference between min , argmin, idf min
  + Min will return the minimum value , can be applied to a column, can’t be applied to a dataframe
  + arg min will return the count of the minimum occurrence (if the minimum value for a column is at position 6, then it returns 5 as count starts from 0), can be applied to columns but not to a whole dataframe
  + idx min will return the corresponding index value for a minimum occurence, can be used on a single column or whole dataframe
* if you want to slice the dataframe or you want to access a particular row based on index, then you can use df.ix
* If you want to get just the minimum value or maximum value for each column or row, then you can pass df[‘col’].min or df[‘col’].max
* if you want to get minimum value of a column, you can use advance methods such as argmax , argmin or idxmax , idxmin
  + sal.loc[sal[‘total\_benefits’].idxmin()]
  + sal.iloc[sal[‘total\_benefits’].argmin()]
* when the index column of a dataframe is having time/date then it is called a **time series index**
* del df[‘day’] # this will remove the day column from df
* pct\_change method for column or rows can be used to calculate percentage change comparing it to previous date
* moving average can be calculated using the below example:
  + bank\_stocks['BAC']['Close'].loc['2008-01-01':'2008-12-31'].rolling(window=30).mean().plot(label='ma')

# matplotlib

* matplot lib is a popular **plotting library** for python
* it gives control over every aspect of a figure or plot
* designed to give a similar feel to matlab’s graphical plotting
* works well with **pandas** and **numpy arrays**
* %matplotlib inline – this will let you see the plots you use on jupyter be visible on jupyter automatically when you try to execute something that is related to plotting information, if you are not using jupyter, then you have to import matplotlib.pyplot as plt, after giving all the code needed for visualization, use plt.show(), this is just like printing the plot

## Plt methods

plt.plot(x,y) #plots 2 arrays or inputs

plt.xlabel(<string that you want to name x axis>)

plt.ylabel(<string that you want to name y axis>)

plt.title(<title that you want to give to the plot>)

### creating single plot on canvas using functions

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np

x = np.linspace(0, 5, 11)

y = x \*\* 2

plt.plot(x, y, 'r') # 'r' is the color red

plt.xlabel('X Axis Title Here')

plt.ylabel('Y Axis Title Here')

plt.title('String Title Here')

plt.show()

### creating fig using OOPS way

when you create figures using OOPS way you can create multiple figures and can also create figures inside figures and give different parameters using their object reference

#### add\_axes method

it will accept 4 parameters,

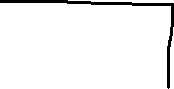
1. starting x point
2. starting y point
3. offset from starting x point (end point (on x axis) horizontal distance from starting x point)
4. offset from starting y point (end point (on y axis) vertical distance from starting y point)

##### functional example

1. here you will create a **figure object fig using plt.figure**
2. then you will create **axes object using fig.add\_axes** method
3. then on this **axes object** we will do all operations such as axes.plot, axes.set\_title, axes.legend, you can pass below parameters such as color and so on
   1. color=<’color name or hex code’> #ex: green or you can give any name you find with crayons- ex: gray, purple, or hexcode such as FF8C00
   2. linewidth = <int or float> or lw = <int> # width of the line
   3. alpha = <value between 0 and 1 (both 0 (full transparent) and 1 (no transparency) included)> decides the transparency of the line
   4. linestyle=<quoted symbol> #which decides the type of plot to be on the line, example values: ‘:’ ,’-‘ , ‘-.’ , ‘steps’
   5. if you want the plotted points of (x,y) then you can pass marker=’o’ which will plot the x versus y mapping on graph
      1. # possible marker symbols: marker = '+', 'o', '\*', 's', ',', '.', '1', '2', '3', '4', ...
   6. markersize=<int> or ms=<int> #which decides the size of the marker for plot of (x,y)
   7. other parameters are markerfacecolor, markeredgewidth, markeredgecolor
4. axes.set\_xlim, axes.set\_ylim : if we have a plot that starts from 0,0 and 5,25 the plot looks something like below

Chart, line chart

Description automatically generated



But if you want only the part that is highlighted with black sketch, then you have to use set\_xlim, set\_ylim using axes object. ex: ax.set\_xlim([3,4]), ax.set\_ylim([10,15])

###### ex:

import numpy as np

x = np.linspace(0, 5, 11)

y = x \*\* 2

fig=plt.figure()

fig.add\_axes([0,0,1,1])

axes1=fig.add\_axes([0.1,0.1,0.8,0.8])

axes2=fig.add\_axes([0.2,0.2,0.4,0.4])

axes1.plot(x,y)

axes2.plot(y,x)

axes1.set\_title(‘axes 1 title’)

axes1.set\_xlabel(‘x- label’)

output

Chart, line chart

Description automatically generated

###### legend

if you are having a figure with multiple plots inside it, then you need to specify detail about each plot, such that it will increase readability

example

fig=plt.figure()

ax=fig.add\_axes([0.1,0.1,0.8,0.8])

ax.plot(x,x\*\*2,label='X-squared')

ax.plot(x,x\*\*3,label='X-cubed')

ax.set\_xlabel('x-axis')

ax.set\_ylabel('y-axis')

ax.set\_title('plotted after understanding concept')

ax.legend(loc=0) #this is needed for the legend to be applied on figure, and loc=0 (there are other loc values which gives different positions to place the legend on figure) means the position of the legend will be displayed on the best suitable positon without disturbing the plot, or you can also pass the string to loc by saying where you want to position the legend, ex: center, center left, upper left link: <https://matplotlib.org/stable/api/legend_api.html#:~:text=a%20numeric%20value%3A-,Location%20String,-Location%20Code>

, else you can pass a tuple as custom co ordinates where you want to list the legend

#### OOPS approach

We can create a figure object using plt.subplots and then create a sub plot and call that subplot object and add as many points to be plotted

##### Multiple axes on same canvas

fig=plt.figure()

ax1=fig.add\_axes([0,0,1,1])

ax2=fig.add\_axes([0.2,0.5,.2,.2])

ax1.set\_xlabel('x')

ax1.set\_ylabel('y')

ax2.set\_xlabel('x')

ax2.set\_ylabel('y')

ax1.plot(x,y)

ax2.plot(x,y)

###### output

Chart, line chart

Description automatically generated

##### Ex using subplots:

fig,axes = plt.subplots(nrows=1, ncols = 2, ,figsize = (15,2) , dpi=100) # this will arrange 1 x 2 = 2 plots in a 1 x 2 matrix form, fig size(x,y) – x,y are width and height, should be in inches, dpi means dots per inch or pixels per inch

# here axes we get is similar to the ones that are created in the fig.add\_axes method, difference between plt.figure and plt.subplots is that plt.subplots does the implicit add\_axes method based on rows and columns

plt.tight\_layout() # this will prevent overlap of plots

axes # here axes will hold a nrows x ncols dimensional array where you can iterate upon (OO method) and store different plot values for different subplots or you can access them using indexing such as index[0]

# plotting using for loop

for current\_axes in axes:

current\_axes.plot(x,y)

# or you can give

axes[0].plot(y,x, label=’ y square rooted’)

axes[0].set\_title(‘plot using OOPS’)

axes[1].plot(x+1,y, label=’x+1 vs xsquared’)

axes[1].set\_xlabel(‘x axes’)

ax.legend(<loc value as discussed above or a tuple with co ordinates to inform matplotlib to where to list it on figure>)

fig.save\_fig(‘mlfig.png’, dpi=100) # you can also specify dpi in the current savefig method, if you forgot to specify it at above code

##### creating sub plots using plt.subplot on same canvas

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import numpy as np

x = np.linspace(0, 5, 11)

y = x \*\* 2

# plt.subplot(nrows, ncols, plot\_number)

plt.subplot(1,2,1)

plt.plot(x, y, 'r--') # More on color options later

plt.subplot(1,2,2)

plt.plot(x,z,linewidth=3,linestyle='\*-',color='green')#another type of parameter declaration

###### output:

Chart

Description automatically generated

### plot methods

you can have the following plots using plt

1. normal graph like plot
2. scatter plot, ex: plt.scatter(x,y)
3. histogram plot - plt.hist(data)
4. box plot - plt.boxplot

### style sheets

Matplotlib has style sheets (http://matplotlib.org/gallery.html#style\_sheets) you can use to make your plots look a little nicer. These style sheets include plot\_bmh,plot\_fivethirtyeight,plot\_ggplot and more. They basically create a set of style rules that your plots follow. I recommend using them, they make all your plots have the same look and feel more professional. You can even create your own if you want your company's plots to all have the same look (it is a bit tedious to create on though).

Ex: plt.style.use('ggplot')

### Matplotlib Misc:

* Line plot: to draw a line plot you can simply use matplotlib.pyplot.plot method or df[‘col’].plot or df.iplot or df.plot, if you use df.plot, each column will be plotted as a separate line and you can have legend information as well

### further reading

* [http://www.matplotlib.org](http://www.matplotlib.org/) - The project web page for matplotlib.
* <https://github.com/matplotlib/matplotlib> - The source code for matplotlib.
* <http://matplotlib.org/gallery.html> - A large gallery showcaseing various types of plots matplotlib can create. Highly recommended!
* <http://www.loria.fr/~rougier/teaching/matplotlib> - A good matplotlib tutorial.
* <http://scipy-lectures.github.io/matplotlib/matplotlib.html> - Another good matplotlib reference.

# Seaborn

* Statistical plotting library
* Bulit **on top of matplotlib (**that is why we will use **%matplotlib inline** in jupyter notebook even for seaborn plot**)**
* Designed to work very well with pandas dataframe objects
* Seaborn comes with builtin datasets that can load instantly
* Ex: tips=sns.load\_dataset('<dataset name>')

## Distribution Plots

### Dist plot (Distribution plot)

* Allows us to show distribution of a univariant (one variable) set of observations
* Need to pass only one variable(a column data) for this plot
* ex: sns.distplot(tips['total\_bill'],kde=False,bins=20)

#### Parameters for distplot

* bins -> how many ranges in which you want to divide your data in to (ex: 0-10,10-20,20-30 …).how many histogram bars do you want

### joint plot

* similar to plotting 2 dist plots at once, urf- allows you to handle bivariate data

#### declaration 1:

here you will specify column names directly and pass dataframe for keyword data

sns.jointplot(x='total\_bill',y='tip',data=tips,kind='kde')

#### declaration 2:

here you can plot a joint plot for dataframes of 2 different columns, by just passing the row or column

sns.jointplot(x=tips['total\_bill'],y=flights['tip'])

#### parameters for jointplot

* x : variable whose values you want to plot on x axis
* y : variable whose values you want to plot on y axis
* data: dataframe whose data you will be plotting on x and y
* kind: type of plot you want to plot on graph
* ex: reg, scatter (default), hex, kde

### pair plot

* plots data for numerical columns in the data in the form of 2 pairs, for example if you have a,b,c as numerical columns, then plotting will be done in the form of following pairs (a,a),(a,b) ,(a,c) ,(b,a) ,(b,b) ,(b,c) ,(c,a) ,(c,b) ,(c,c) and accepts categorical data as hue (which will let each category being represented with a unique color on the plot, below you can see males and females getting displayed with different colors)
* size of dataset is parallel to the execution time
* ex: sns.pairplot(data=tips,kind='scatter',hue='sex',palette='coolwarm',) #here palette accepts a wide variety of values which you can choose from when you google **matplotlib colormap**

Chart, scatter chart

Description automatically generated

#### parameters for pairplot

* data : data that you want to be plotted
* kind: type of plotting mechanism
* hue: categorical column that you want to depict differently in plots

## Categorical Plots

### Bar plot

Visualization by performing action by grouping them into categories, here the action can be any aggregation function (inbuilt or custom)

Ex: average tip by male and female- first group by sex and then calculate average tip for each of them

Example syntax:

import numpy as np

sns.barplot(x='sex',y='total\_bill',data=tips,estimator=np.std)

### count plot

similar implementation to bar plot, but the action here is count of occurrence, so there is no need of second variable

ex: sns.countplot(x='sex', data=tips)

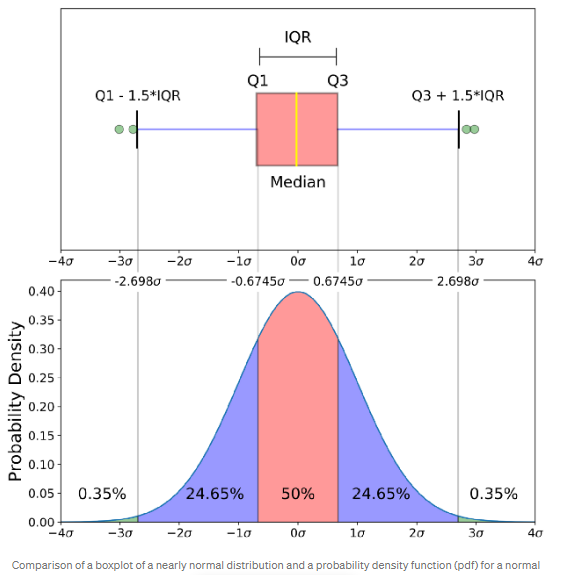
### box plot:

plotted between a categorical and numeric column, can also add a hue.

sns.boxplot(x='day', y='total\_bill', data=tips, hue='smoker')

Chart, box and whisker chart

Description automatically generated



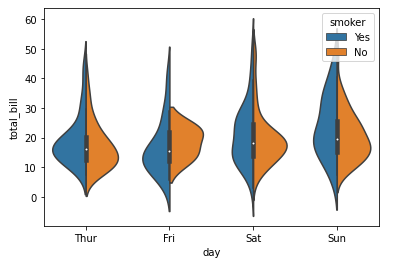
### Violin plot

Similar to box plot, has good amount of data in form of visualization, but difficult to visualize as plot is a bit ambiguous

Ex: sns.violinplot(x='day',y='total\_bill',data=tips,hue='smoker', split=True)

Hue= smoker will let bill be calculated in 2 parts: smoker – yes or no

Split=True will let the hue be displayed on 2 sides



### Strip plot

Scatterplot for categorical values based on numerical values

Ex: tip given on any day of a week

Problem here is, for repeating patterns, overlap happens and we cant know the intensity of the overlap, to avoid that we will set jitter=True

Ex:sns.stripplot(x='day',y='total\_bill',data=tips,hue='sex',dodge=True,jitter=True)

Chart, scatter chart

Description automatically generated

### swarm plot

A mixture of scatterplot and violin plot (and is similar to box plot), except that all points (maximum) are plotted on the graph

Won’t work well with huge data

Ex: sns.swarmplot(x='day',y='total\_bill',data=tips)

Chart, scatter chart

Description automatically generated

### Factor plot

You can pass the data you want and also you can specify the type of plot you want, it basically means you can specify the type of plot at runtime, types of plots supported are "strip", "swarm", "box", "violin", "boxen", "point", "bar", or "count"

Ex: sns.catplot(x='day',y='total\_bill',data=tips,kind='box')

### kdeplot

syntax: sns.kdeplot

## Matrix plots

To perform matrix plots need we need matrix form i.e., names for index and column names should indicate value or hold information ex: df.corr(), df.pivot\_table

### Heat plots

Heatplot show relation between an index values and column values

Syntax: sns.heatmap(<matrix plot>,annot=<bool>,cmap=<type of coloring you want>,linecolor=<>,linewidth=<>)

Note: in the below example you can notice total\_bill, tip, size (as rows) is plotted against total\_bill, tip, size (as columns) which represents that it needs matrix form

Chart

Description automatically generated

Ex2:

fp=flights.pivot\_table(index='year',columns='month',values='passengers')

sns.heatmap(fp,cmap='magma',linecolor='yellow',linewidth='2')

Background pattern

Description automatically generated

#### Heatmap using iplot

closing\_price\_corr.iplot(kind='heatmap', colorscale = 'rdylbu')

### Cluster map:

This tries to show columns and rows of map together based on similarity, and clusters based on th index values passed through the df (which is in matrix form)

As clustermap might be having multiple values standard\_scale = n will let values be standardized between 0 and n

Ex: sns.clustermap(fp,cmap='magma',standard\_scale=1)

Chart, bar chart

Description automatically generated

## Grids

Used to automate subplots based on features of data

### PairGrid

Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.

We can compartmentalize grids to plot different plots, or plot same plots across all grids

In this we can compartmentalize grids in to

1. upper (cells above diagonal)
2. diagonal
3. lower (cells below diagonal)

#### same plot across all grids

g = sns.PairGrid(iris)

g.map(plt.scatter)

#### compartmentalize grids

g = sns.PairGrid(iris)

g.map\_diag(plt.hist)

g.map\_upper(plt.scatter)

g.map\_lower(sns.kdeplot)

### FacetGrid

FacetGrid is the general way to create grids of plots based off of a feature

Facet grid is useful when you want to check plot based on certain condition, for example you want to plot different histograms of **total\_bill** for a **smoker** and **nonsmoker** for **different member counts**

Syntax:

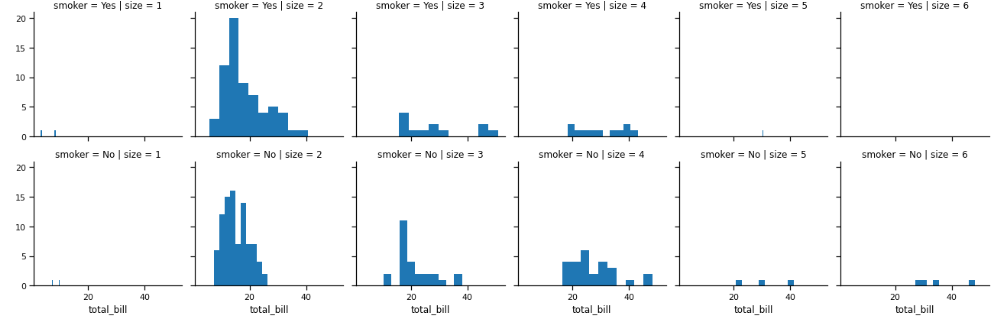
g=sns.FacetGrid(data=<df>, col =<>, row = <>)

g.map(<plot type>,<column1>,<column2>..)

ex:

g=sns.FacetGrid(data=tips,col='size',row='smoker')

g.map(plt.hist,'total\_bill')



#### Note:

We have given col= size, size has categorical values of 1 to 5, so we can see 5 columns in above plot

Row=’smoker’ smoker has 2 categories, therefore we will have 2 rows of projecting the same, when you map it to a plot, using p.map(plt.hist,’total\_bill’) you are plotting it as histogram considering total\_bill to be plotted as a univariant plot

## regression plots

### lm plot

helps to plot linear models with seaborn

ex: sns.lmplot(data=tips,x='total\_bill',y='tip',col='smoker',row='time',hue='sex',markers=[ 'x','v'],aspect=0.5,height=8)

note: having row and column parameters will make the plot look like facet grid, you can also specify markers

## Style and color

### sns.set\_style

to set background

### sns.despine

used to remove or set border lines of graph

### size of figure

plt.figure(figsize=(12,3))

as sns uses matplotlib under the hood, we use figsize to set size

### sns.set\_context(context=<poster or notebook or paper>, font\_scale=<int>)

this helps in modifying plot diagram as per the need, also to set fontsize

# Plotly and cufflinks

* plotly is an interactive data visualization library
* cufflinks connects plotly with pandas
* plotly is completely free in offline mode to handle visualizations and if you want to handle vizualisations online, then you need to go for paid options
* you need to do a set of installations before using plotly
  + !pip install plotly
  + !pip install cufflinks
  + !pip install chart-studio
* Few imports to be done before hand
  + import plotly,cufflinks as cf
  + import chart\_studio.plotly as py
  + from plotly.offline import download\_plotlyjs,init\_notebook\_mode,plot,iplot #plotly.js is used so that it can create iplot (interactive plots )
* custom parameters to be set
  + init\_notebook\_mode(connected=True) # will connect JS to notebook, so that we can see everything in the notebook
  + cf.go\_offline() #allows us to use cufflinks in offline mode

## iplot

an interactive way of plotting where you can hover over the plot (to get details), zoom in or out, save as png, remove or add particular parameters

### scatterplot

df.iplot(kind='scatter',x='A',y='B',mode='markers',size=5)

### barplot

df.iplot(kind=’bar’)

A picture containing needle, chime, writing implement, pencil

Description automatically generated

df.sum().iplot(kind=’bar’)

we have used an aggregate function here because bar plot in general needs values which don’t fluctuate much, If they fluctuate too much, there is nothing left for in sights, so using an aggregation function and the plot will be as follows

Chart, box and whisker chart

Description automatically generated

### Box plot

No need for implementing aggregate functions like we did in bar plot, because box plot implements aggregate functions under the hood

df.iplot(kind=’box’

### 3d plot

df3=pd.DataFrame({'x':[1,2,3,4,5],'y':[100,200,500,300,400],'z':[5,4,3,2,1]})

df3.iplot(kind='surface',colorscale='rdylbu')

### hist plot

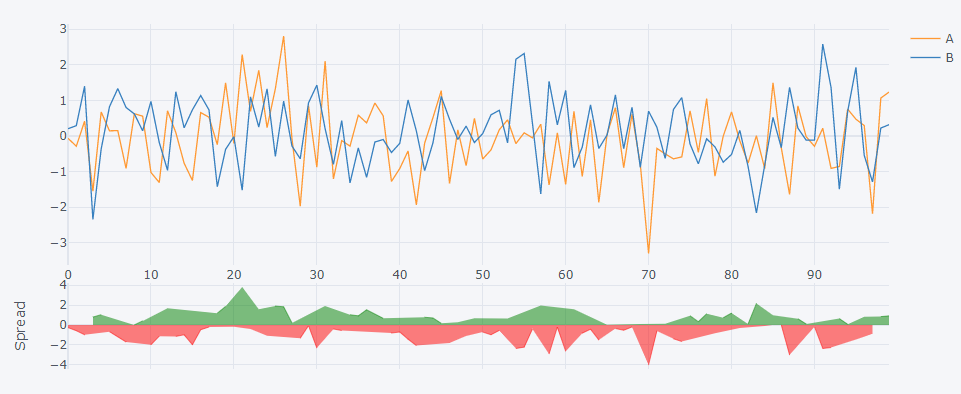
ex1: df['A'].iplot(kind='hist',bins=20)

ex2: df.iplot(kind='hist',bins=20)

### spread plot

used to compare stocks data

df[['A','B']].iplot(kind='spread')



### Bubble plot

This is similar to scatter plot, but here you can decide the size of a particular bubble (x,y) based on the third column value z

df.iplot(kind='bubble',x='A',y='B',size='C')

in the above example you are taking A and B columns as X and Y, taking size values as corresponding C column entry.

Used for examples of world GDP, happiness index

### Candle plot

BAC[['Open','High','Low','Close']].loc['2015-01-01':'2016-01-01'].iplot(kind='candle') #you need to pass Open, Low, High, Close as they are the only values needed to plot candles

### Technical analysis plots ta\_plot

#### EMA and SMA

Ema:

bank\_stocks['MS']['Close'].loc['2015-01-01':'2015-12-31'].ta\_plot(study='ema',periods=[13,22,51]) #you need to pass a particular value such as open or low or high or close

SMA:

bank\_stocks['MS']['Close'].loc['2015-01-01':'2015-12-31'].ta\_plot(study='sma',periods=[13,22,51]) #you need to pass a particular value such as open or low or high or close

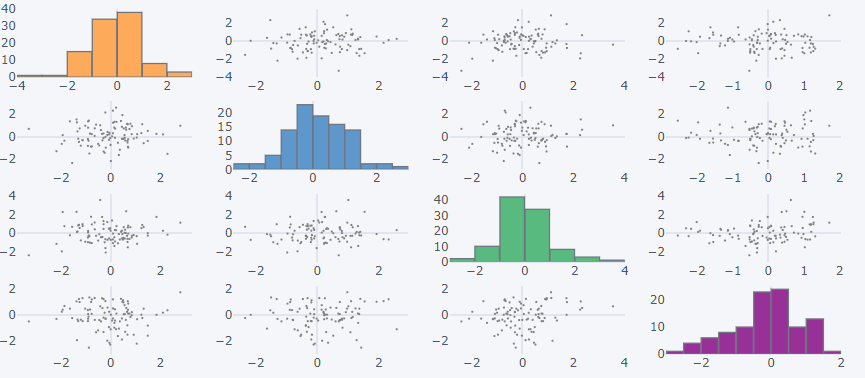
#### Bollinger bands

bank\_stocks['BAC']["Close"].loc['2015-01-01':'2015-12-31'].ta\_plot(study='boll')

#you need to pass a particular value such as open or low or high or close

### Scatter matrix

Scatter plot in similar to pair plot representation, if tried for large datasets, might crash python kernel



# Geographical plotting

* geographical plotting is difficult because of the reason that various different type of data is to be used for plotting
* using plotly for plotting is a good choice because it has a basemap extension which will help in generating static geographical plots
* it looks a bit typical as the data passed to it should be compatible, for example a location parameter below should have abbreviated country names which are to be in plotly supported form

## plotting USA

example code:

import chart\_studio.plotly as py

from plotly.offline import download\_plotlyjs,init\_notebook\_mode,plot,iplot

import plotly.graph\_objs as go

init\_notebook\_mode(connected=True)

#

data=dict(type='choropleth', #specifies what type of geographical plot we are doing

locations=['AZ','CA','NY'], #list of state abbreviation codes

locationmode='USA-states', #county level

colorscale='Jet',

reversescale = True # this will let you have the colour scale reversed(along with values) so you can decide wether to have a thick color or thin color for a column (you want to plot values of) value which is high

text=['Arizona','California','NewYork'],#text which we want when we are hovering on plot, for above locations element wise

z=[1.0,2.0,3.0],#actual values that are going to be shown on colorscale and plot above mentioned locations with these color codes

marker = dict(line = dict(color = 'rgb(255,255,255)',width = 2)), #the line parameter will let the states in the USA get seperated by line

colorbar={'title':'colorbar title goes here'})

layout=dict(geo={'scope':'usa'})

choromap = go.Figure(data = [data],layout=layout) # we need 2 objects- layout, data

iplot(choromap)

* it is almost impossible to remember the syntax as it is a bit complex, it is wise to refer documentation and also we need 2 variables **data** and **layout** with appropriate values to plot

## world plot

data=dict(type='choropleth',

locations=world['CODE'],

locationmode='country names' # this is needed if the above locations is not accepting code (for example Afghanistan-> AFG) as its series(a column of dataframe) value for above **locations** parameter.

z=world['GDP (BILLIONS)'],

text=world['COUNTRY'],

colorbar={'title':'GDP in billion USD'},

)

layout3=dict(title='Global GDP',

geo=dict(showframe=False,

projection={'type':'stereographic'}#type of projection you want on graph

)

)

choromap3=go.Figure(data=[data],layout=layout3)

iplot(choromap3)

# Anaconda

Anaconda is a popular distribution (an implementation of language along with tools and (or) libraries) of python

Jupyter: a popular development environment in anaconda where you can run code, display images, write markdown code within a single notebook

* To update python in conda env: conda install python= <version you want to update>
* **env** command will list all environment variables in linux

-You can manually specify the path to the conda executable to use for activation (version 4.4+). To do so, open the Command Palette (Ctrl+Shift+P) and enter Preferences: Open User Settings. Then set python.condaPath, which is in the Python extension section of User Settings, with the appropriate path.

-we can have different environments in conda, and each environment can have different packages installed.

## to activate a particular environment we have to use

conda activate <environment name>

## similarly to deactivate a particular environment we have to use

conda deactivate <environment name>

## to remove existing package:

conda env remove -n <name of package >

## to see the list of already available conda environments

conda info --envs

(snowflakes) saivinil\_pratap@TIGER02143:~$ conda info --envs

# conda environments:

#

base /home/saivinil\_pratap/miniconda3

snowflakes \* /home/saivinil\_pratap/miniconda3/envs/snowflakes

the \* shows that the snowflakes is the current active environment,alternatively you can confirm this by seeing the name that is in the brackets above when we try to execute the command "conda info --envs"

## if you want to move back to base environment, you can simply type-

conda activate

## create new environment

### if you just want to create environment

conda create --name <environment name>

### if you want to specify python version and list of packages you want to install in this environment

this will move the environment to default environment which is base.

-when you create a new environment conda uses the same python version which is used to download and install anaconda, if you want to create a new environment with different python version, then you have to use the following command (conda env create or create env):

conda create --name <environment name> <packages you want to install, separated by space> python=<version>

ex: conda create --name MLproj-env numpy scipy python=3.7

## creating conda environment from existing yaml or another file

execute the below command

conda env create -f <path of the environment file>

## creating new environment from existing environment

Yes, there is a conda command to create a duplicate environment from an existing environment. The command is conda create with the --clone option followed by the name of the environment you want to clone.

Here's an example command:

conda create --name new\_environment --clone existing\_environment

## renaming existing\_environment

conda rename -n <old name> <new name>

## to delete conda environment:

conda env remove -n <env name>

## if you want to search for a package in ANACONDA REPOSITORY

ex: conda search \*eauti\* (it supports wild card entries and it will check the repository and lists the package that has "eauti" inside it )

## Export your active environment to yml file

conda env export > <path of the file where we want to save env file>

## exporting your current environment to yml file without build information

conda env export --no-builds > <environment file name>

ex: conda env export --no-builds > env.yml

## updating current conda environment using yaml file

conda env update --file <yml file location>

## conda lock

used to lock the dependencies and

## .condarc file

Can be abbreviation of **CONDA** **R**untime **C**onfiguration or **CONDA** **R**eusable **C**onfiguration.

This is a yaml configuration file which will have the conda preferences.

You can edit the .condarc file using any text editor.

Here are some of the things that can be stored in the .condarc file:

* Channels: A list of channels to search for packages.
* Package priorities: The priority of each channel in the search order.
* Proxy settings: The proxy server to use for network requests.
* Environment directories: The directories to search for environments.
* Prompt: The text to display in the shell prompt.
* Other configuration options: There are many other configuration options that can be set in the .condarc file. For a complete list, see the conda configuration documentation.

## Updating all packages in current conda environment

conda update –all

pip3 list -o | cut -f1 -d' ' | tr " " "\n" | awk '{if(NR>=3)print}' | cut -d' ' -f1 | xargs -n1 pip3 install -U

pip install pipupgrade

pipupgrade --verbose --latest --yes

# Pip commands

## pip environment related

* Pip virtual environment
  + To create environment using pip:
    - python3 -m venv <env-name>
  + after you create environment it will create a folder **with the name of the environment you gave in python3 -m venv <env-name>** in the directory where you ran the command, and it will store all related info in the folder, so to activate the environment in the below command you will call **activate w**hich is stored in the folder that is created,
  + To activate pip environment in linux (should be done from the folder where it is created)
    - source <env-name>/bin/activate
  + else if you created the environment in the root folder and you are somewhere deep inside another folder, you can give
    - source ~/<environment-name>/bin/activate
  + to deactivate environment
    - type deactivate

### if you are not able to detect installed python environment

if you have installed a python package and then not able to run it by executing a python program, but when you are trying to install, it says “requirement satisfied” and you are also able to import it in python interpreter, you can do

python

import <package>

package.\_\_path\_\_

this will give you the lib path, for example '/home/tiger03104/miniconda3/envs/experiment/lib/python3.11/site-packages/poetry'

now you at lib level of the path you will find a bin, and using that bin path (or simply replace the lib in above path with bin (provided that bin folder exist at that level) and omit the rest of the path, example – ‘/home/tiger03104/miniconda3/envs/experiment/bin’ and add that to path

#### note

sometimes even after adding path, you might not be able to import, so recheck after you add path to your path and restart and check even if still things don’t work.

## pip install -e .

The command **pip install -e .** is used to install a package in "editable" or "developer" mode. The **-e** or **--editable** flag is used to specify that the package should be installed in editable mode. The **.** at the end of the command is used to specify the current directory as the location of the package to be installed.

When you run the command **pip install -e .**, pip will install the package in the current directory in editable mode. This means that changes made to the package's source code will be immediately reflected in the installed package, without the need to re-install it. This is useful for development and testing, as it allows you to easily test changes to the package without having to go through the full installation process each time.

It's worth noting that the package to be installed must have the setup.py file in the current directory, and the package should be in a structured format, following the best practices and conventions, before being installed in editable mode.

Also, when installing in editable mode, pip will not create a new virtual environment, and it will install the package in the current environment. So make sure that you are in the correct environment where you want the package to be installed.

## Exceptions and assertions in python

### Default Exceptions:

--raise <error\_name> will raise an error ,if in try block will turn to exception block stopping execution in try block,else raises an error and stops execution

ex:raise ValueError("any message you want to type for that error"),message is optional

#### type 1:

try:

raise MemoryError("this is error",'second error')

except MemoryError as e:

print(e.args)

#### type 2:

try:

raise MemoryError

except:

print("any error message you want")

#### type 3:

try:

raise

except:

print("error raised manually")

#### type 4:

try:

raise MemoryError("this is error")

except Exception as e:

print(e)#incase you the chance of having one of the many exceptions getting raised. it will print the message that is typed with exception

--

import traceback#python traceback will help us get lot of information about how an exception is raised

try:

raise Exception('this is just to get an implementation of traceback')

except:

error\_file=open(r"C:\Users\saivinil.pratap\Desktop\log.txt",'a')

error\_file.write(traceback.format\_exc())

error\_file.close()

### custom exceptions

class error(Exception):

"""base class for other esceptions"""

class invalid\_length(Exception):

"""when you pass invalid number of columns"""

try:

raise invalid\_length

except invalid\_length as e:

print("you need to pass valid number of arguments")

### assert in python:

l=-1

'''assert is more or less like exception in python, to explain it with an example:

if you want to have only positive number in list and if you get any negative number you need it to be informed

in those cases you use assert statements

when the assert statement is false the string next to it will be raised as an assertion error

here we don’t have try and except so when an assert statement is failed it will stop there itself, assert statement is for programmers errors while programming and these statements can be removed after the application is fully developed as user side issues will be dealt with exceptions

'''

assert l>0, 'number is not positive'

## different types of installation ways to use from inside the python script (.py file)

### check and install a package if it does not exist

* to install a package inside script (installing package inside script) subprocess check\_call: A call to this function runs the command with the arguments, waits for it to complete, then gets the return code. If zero, it returns, else it raises CalledProcessError. Such an object holds the return code in the returncode attribute.

spec = importlib.util.find\_spec('sagemaker')

if spec is None:

subprocess.check\_call([sys.executable, "-m", "pip", "install", "-U", " sagemaker==2.24.1"])

* + the above code checks whether the sagemaker 2.14.1 is installed or not, if not it will install, if yes does nothing

### method 2

import sys

!{sys.executable} -m pip install "sagemaker>=2.99.0"

#this will check whether the listed version is installed, if not, interpreter will install

### Method 3

Installing multiple packages and upgrading them in one line

! pip install botocore boto3 awscli --upgrade

## Write a python file from jupyter cell

%%writefile <path of the python file>

<lines of python code that you want to store as a .py file>

The above lines of code will create a python file in the above specified path

## reload a package (useful if we recently updated a package)

* If you make a change to a module and need to reload it, you need to either restart the interpreter or use a function called reload() from module importlib
  + syntax: importlib.reload(<package name>)

## Normal distribution

A function that represents the distribution (distribution means how values are distributed for a field) of randomly generated variables to represent a bell shaped curve

### Random distribution generation and plot using python

Code:

df=pd.DataFrame(np.random.randn(1000))

df.plot.kde()

diagram representing above code:

A picture containing text, plot, line, diagram

Description automatically generated

## Python code to play sound

!pip install chime

import chime

chime.theme('zelda')

chime.success() #sound 1

chime.warning() #sound 2

chime.error() #sound 3

chime.info() #sound 4

chime.notify\_exceptions() #sound 5

# string\_func:

-- to convert a timestamp which in in string to timestamp we need to use string.strptime to convert it into a timestamp format

--string.rsplit(separator, maxsplit)

Parameter Values

Parameter Description

separator Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator

maxsplit Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

--string.replace(old, new, count)#we can replace as manytimes as we want look at the count parameter

--if you want a string with 5 dollar symbols just give stri=5\*'$'

--shorthand assignment operators are also called as augmented operators

--fstrings this is one of the best way to include variables in string the easier way, you just need to place f before the string and use flower braces for every variable you want to include in the string

t='temp'

stri=f'{t}hi' #this is valid and holds temphi

-- to withhold a large strings we can use '''(triple quotes) the information that is hold between triple quotes can be assigned to a variables

--in python strings we have escape sequences those escape sequences will start with '\' but there might be instances where we want the compiler to treat the '\' as a back slash , not an escape sequence, in that scenario preceeded by r so it will be treated as raw string.

raw\_str=r'this is a raw string which treats back slash as a '\' not as a escape sequence'#it also ignores escape sequences

raw\_str will be stored differently and will be printed(sent to console) differently

ex:

raw\_str=r'this is a raw string which treats back slash as a \\'' not as a escape sequence'

print(raw\_str)

raw\_str

--<string>.islower function will check wether all the elements in the string are lower or not and then return true or false accordingly, same happens with isupper function.

ex:

'123'.isupper()

'123'.islower() #both return false

'A12'.isupper() #true

'a12'.islower() #false

some similar functions are isdigit,isalnum

--<string>.isspace()#will return true if string is containing only space characters

ex:' \t'.isspace()#true

ex:' '.isspace()#true

ex: ' \b'.isspace()#false the string should have only blank spaces but \b is a blank space

--<string>.isdecimal()#will return true if the number is decimal

ex:'11'.isdecimal()#will return truw

,'11.1'.isdecimal()#will return false

,"\u0030".isdecimal(),"\u0047".isdecimal()

'\u0030'->unicode for '0'#will return true

'\u0047'-> unicode for 'G'#will return false

--<string>.istitle()# will return true if the starting letter of the text is capital, else letters in the first word should not be.

ex:

'Dad name is Pratap Badri Narayana'.istitle()#true

'dad'.istitle()#false

'DaD'.istitle()#false will return true only if the first letter (not other letters) in the first word are capital

--rjust and ljust will add spaces on the right side and left side making it the required length.

<string>.rjust(integer,pattern) or <string>.ljust(integer,pattern) or <string>.center(integer,pattern)#pattern is optional

ex:'hello'.rjust(10)#' hello' #rjust means keeping the text at rightside

ex:'hello'.ljust(10,'$')#'hello$$$$$' #ljust means keeping the text at leftside

ex:'hello'.center(10,'$')#'$$hello$$$'#center means keeping the text at center

note: rjust,ljust,center can be used recursively

ex:'hello'.rjust(10,'l').ljust(15,'r').center(20,'y') #will first append l at left to hello, then append r to right of hello, then append y to both sides of the result string, recursion is possible because each rjust,ljust or center will return a string.

--starts with and ends with are python string functions which will check wether the pattern is matching with the input string if matches will return true else false.

ex:'sadefcnedkfjcndkc'.endswith('dkc')#true

--strip function will remove the specified character at the both ends as long as the character pattern exists,whereas rstrip only removes at right end and lstrip removes only at left end

ex:'ppdfppppppppppppppppp'.rstrip('p')#'ppdf'

--strip function tries to strip letters(each seperately) that are passed as a parameter

ex:'abcabcdefabcghiabccab'.strip('abc')

the output is 'defabcghi' as it strips a or b or c that esist at the starting or at the end.

--when we want to concatenate multiple strings as a single string, we can use string formatting

ex:

name,age,sex,country='vinil','21','Male','india'

details='my name is %s, my age is %s, i am a %s, i am from %s' %(name,age,sex,country)#%s used here is called conversion specifiers

details

output:'my name is vinil, my age is 21, i am a Male, i am from india'

--import textwrap

string = "T his i ss a verry verry very very very long string."

print (textwrap.fill(string,4))

print( textwrap.wrap(string,4))

T #have to play with it and check

his

iss

a ve

rry

verr

y

very

very

very

long

stri

ng.

['T', 'his', 'iss', 'a ve', 'rry ', 'verr', 'y', 'very', 'very', 'very', 'long', 'stri', 'ng.']

###############################################

# math\_library:

math.sin(value) will calculate the sine angle of the value in radians

math.degrees(radians) will calculate values in degrees

math.hypot(value1,value2) will calculate hypotenuse

###############################################

# eval\_functionality:

it only acccepts string as its input.

eval is a very powerful function built in python, it helps in evaluationg an expression, an expression can be a statement or an code object

eval(polynomial expression in string format) will return the value by replacing variables with values that are defined by us

it will also take inbuilt functions or variables or keywords or defined functions. ex:type(eval('int'),type(eval('len'))

ex:

x,y=1,2

eval('x\*\*3 + x\*\*2 + y + 1') #will return 5 as output which is equal to the calculation of replacing variables in expression with values

ex:a=[1,2,3,4,5]

eval('a.insert(3,5)')

a#[1, 2, 3, 5, 4, 5]

--polynomial can be calculated in the below manner as well, this manner is useful when we want to create a polynomial easily without manual intervention

a list of values can be converted into a polynomial by using numpy.poly1d(list)

ex1:

import numpy as np

p1=np.poly1d([4,9,5,-4])

#p1 will be of datatype numpy.poly1d and the expression will be 4x\*\*3+9x\*\*2+5x-4

#if you want to pass the x value as 2 then

c=p1(2) # this will pass 2 as value of x to the above polynomial

ex2:

d=np.polyval(np.poly1d([4,9,5,-4]),np.poly1d(2))

#value of c and d[0] will be same in both the examples

--eval() will return int and bool types

ex:eval('False'):-output False-bool type

eval = eval(input("Enter any number of your choice"))

print(eval)

print(type(eval))

Enter any number of your choice: 10 + 10

20

<class 'int'>

###############################################

# regex\_func:

--often we use raw strings in the regex because we might have back slashes and compiler should not misunderstand them for escape sequences

re module has 3 string functions

1:split-> re.split->split(pattern, string, maxsplit=0, flags=0)

normal split -> string.split(separator,count) will split the string using seperator for the given count and returns the output in form od list

ex:'a b c d'.split(' ',2)

2:sub-> re.sub(pattern,substitute,string,count) #replaces the substitute in the pattern till the splitting count is matched

3:subn-> same as sub but returns tuple with the count of substitutions it done

4: re.findall ->

ex:import re

txt = "The rain in Spain"

x = re.findall("ai", txt)

print(x)#['ai', 'ai']returns a list, it returns the elements in the order it is found

5:re.search(): returns the match object(provided that a match is found, if not found it will return None) and it has many methods such as

ex: import re

x=re.search('there','hi there')#x.span() will return the start and end points for first occurence, x.string will return the string that is passed to perform operation on, in the above example it is "The rain in Spain",x.group() returns the part of the string where there was a match

print('match started at ',x.start(),'position', 'ends at ',x.end())#returns None if no match found

6:

symbols used in regex:

--[arn]-> returns match if it contains a or r or n

ex:

import re

txt = "The rain in Spain"

#Check if the string has any a, r, or n characters:

x = re.findall("[arn]", txt)

print(x)

if x:

print("Yes, arn there is at least one match!")

else:

print("No match")

#o/p:['r', 'a', 'n', 'n', 'a', 'n']

--[a-n] -> returns match if it contains letters that are in between a to n INCLUDING a and n

ex:

import re

txt = "The rain in Spain"

#Check if the string has any a, r, or n characters:

x = re.findall("[a-n]", txt)

print(x)

if x:

print("Yes, arn there is at least one match!")

else:

print("No match")

#o/p:['h', 'e', 'a', 'i', 'n', 'i', 'n', 'a', 'i', 'n']

txt="The rain in Spain"

--[^arn] Returns a match for any character EXCEPT a, r, and n

--[0-5][0-9] Returns a match for any two-digit numbers from 00 and 59

--[a-zA-Z] Returns a match for any character alphabetically between a and z, lower case OR upper case

--[+] IN SETS, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

--^(starts with) ->x=re.search('^the',txt) #will return re.match object and you can use it in if condition to check like given below

if(x):

print('match found)

else:

print('doesn't match')

. (denoted by fullstop and fills in any character as replacement) a dot is like a fill in the blank with one space

.{3}-> exactly 3 empty blank spaces which will accept anything as input

.\* -> unlimited blank spaces

abcd$ -> will check wether the string ends with "abcd" or not

[a-m] -> will allow one letter which is in between a and m-%d

\ ->\d signals a special sequence and it will accepts one digit

$(ends with) -> the string before dollar symbol if it matches with the input it is a match

\* -> "zero" or more occurences

+ -> "one" or more occurences

| -> either or

() -> capture and group

special sequences:(these examples are underdeveloped refer link(https://www.w3schools.com/python/python\_regex.asp) if you want extensive information)

--\A ->starts with

ex:

import re

txt = "The rain in Spain"

#Check if the string starts with "The":

x = re.findall("\AThbe", txt)

print(x)

if x:

print("Yes, there is a match!")

else:

print("No match")

--\b -> returns a match if the specified characters are at the beginning of a word or at the ending of a word

ex1 for beginning of a word:

import re

txt = "ain in Spain"

#Check if "ain" is present at the beginning of a WORD:

x = re.findall(r"\bain", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

ex2: if the pattern is present anywhere anynumber of times

import re

txt='ain the rain in spain'

res=re.findall(r'ain\b',txt)

if res:

print(res)

else:

print('no match')

--\B -> it will ignore the match(if exists ) at the start of the string or at the end of the string based on the way it is declared.

ex1(ignores pattern at the start of the string):

import re

txt = "ainThe rin Spain"

#Check if "ain" is present, but NOT at the beginning of a word:

x = re.findall(r"\Bain", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

#the output will contain one 'ain' even though it has two ain's in the string

--\d Returns a match where the string contains digits (numbers from 0-9)

ex:"\d"

--\D Returns a match where the string DOES NOT contain digits

ex:"\D"

--\s Returns a match where the string contains a white space character

ex:"\s"

--\S Returns a match where the string DOES NOT contain a white space character

ex:"\S"

--\w Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore \_ character)

ex:"\w"

--\W Returns a match where the string DOES NOT contain any word characters

ex: "\W"

--\Z Returns a match if the specified characters are at the end of the string

ex:"Spain\Z"

--the \d,\D,\w,\W,\s,\S .... etc are called shorthand codes for character classes, you know the meaning of class , you know the meaning of character,combine them both you get what is meant by character class

--import re

fetchnumberregex=re.compile(r'\d{2}-\d{10}')#the input should be a raw string

first\_number=fetchnumberregex.search('my home number is +91-8332010393 my office number is +92-8072351904')#it will only fetch first occurence of the pattern

all\_numbers=fetchnumberregex.findall('my home number is +91-8332010393 my office number is +92-8072351904')#it will only fetch all occurences of the pattern

print(first\_number.group(),all\_numbers)

output-91-8332010393 ['91-8332010393', '92-8072351904']

--import re

fetchnumberregex=re.compile(r'((\d{2})-(\d{10}))')#the input should be a raw string

first\_number=fetchnumberregex.search('my home number is +91-8332010393 my office number is +92-8072351904')#it will only fetch first occurence of the pattern

all\_numbers=fetchnumberregex.findall('my home number is +91-8332010393 my office number is +92-8072351904')#it will fetch all occurences of the pattern

print(first\_number.group(2),all\_numbers)

output-8332010393 [('91', '8332010393'), ('92', '8072351904')], here the output is different from the above code because we used groups with brackets here and this might be useful for specific use cases

--batRegex=re.compile(r'(bat) (man|mobile|van)')

res1=batRegex.search('bat mobile lost a wheel')

res2=batRegex.search('bat chan is not an valid input')

res1.group(0),res1.group(1),res1.group(2),res2#the 0th index of the match object(example res1)will have the matched object and 1st and 2nd index and so on will have the matching characters that are included in the braces,res2.group will return an error because it will have a None object which doesnot have a group method

--mobregex=re.compile(r'(\d{3}-\d{3}-\d{4},?)')

mobregex.findall('123-123-1231,123-123-1232,123-123-1233,123-123-1234,123-123-1235,123-123-1236,123-123-1237')

mobregex.search('123-123-1231,123-123-1232,123-123-1233,123-123-1234,123-123-1235,123-123-1236,123-123-1237').group()

--haregex=re.compile(r'(ha){3,5}')# this will accept minimum 3 ha's->'hahaha' to 5 ha's ->'hahahahaha' here minimum field or maximum field is optional, here if it has 'hahaha'(3 ha's) and 'hahahahaha' (5 ha's) it will go for 5 ha's this is called greedy approach(greedy means maximum possible value),to get non greedy match check the below example ,if both are left optional then it means there is no minimum or maximum count

haregex=re.compile(r'(ha){3,5}?')#notice the question mark '?' at last

haregex.search('hahahahaha hahaha')#here even if the minimum match(3 ha's->'hahaha') is at last it will be preferred and this is the non greedy approach and this have question mark at last in the re.compile method

ha=haregex.search('hahaha haaaa ha haha hahahaha')

--phoneregex=re.compile(r'((\d{3})-(\d{3}-\d{4}))')

phoneregex.findall('ph 1: 123-123-1234 , ph2: 123-123-1235')

output-[('123-123-1234', '123', '123-1234'), ('123-123-1235', '123', '123-1235')]

the output is in the form of a tuple of lists, each tuple will have the length which is equal to the number of braces we use,the order of items will have from outer bracket to innermost bracket and the order is left to right,i.e,in the re.compile the outer most bracket holds (\d{3})-(\d{3}-\d{4}) the next outermost bracket from left to right has (\d{3}) and the left outermost bracket is (\d{3}-\d{4}) the output follows the above explained order.

--re.compile(r'\d+\s{1}\w+').findall('12 drummers drumming 11 pipers piping') or --re.compile(r'\d+ \w+').findall('12 drummers drumming 11 pipers piping')

#will output ['12 drummers', '11 pipers'] and we have used \s{1} because a space can be a single space or a tab space here we need a single space, a single space can be expressed as normal space (single press on space bar) we use on a english text, it can be understood by the second example code explained above.

--singlevowelsregex=re.compile(r'[aeiouAEIOU]')

couplevowelsregex=re.compile(r'[aeiouAEIOU]{2}')

consonantsandspecialcharactersregex=re.compile(r'[^aeiouAEIOU]')

print('to return single vowel characters we use "[aeiouAEIOU]"',singlevowelsregex.findall('robocop eats baby food'))

print('to return couple vowel characters we use "[aeiouAEIOU]{2}"',couplevowelsregex.findall('robocop eats baby food'))

print('to return consonants characters we use "[^aeiouAEIOU]" caret symbol means everything expect characters specified in regex',consonantsandspecialcharactersregex.findall('robocop eats baby food'))

-- import re

>>> startswithregexobj=re.compile(r'^hello')

>>> startswithregexobj.search('hello there')

<re.Match object; span=(0, 5), match='hello'>

>>> endswithregexobj=re.compile(r'there!$')

>>> endswithregexobj.search('hi there!')

<re.Match object; span=(3, 9), match='there!'>

>>> allnumbersregex=re.compile(r'^\d+$')

>>> allnumbersregex.search('1223456789')

<re.Match object; span=(0, 10), match='1223456789'>

>>> allnumbersregex.search('12234x56789')

>>> allnumbersregex.search('12234x56789')==None

True

note:here in the above example allnumbersregex(which will use both ^ and $) will not accept any other character than number because the regex explains it should have one or more numbers which inturn can indicate that more than one = length of the input passed

--the '.' character class will act as a placeholder for any character except newline i.e, a digit or a number or a space is accepted.

ex:>>> atregexobj=re.compile(r'.{1,2}at')#a flower bracket indicates a definite number,but here it varies and allows a length of 1 or 2

atregexobj.findall('the cat along with the bat are lying on a flat mat')

output:[' cat', ' bat', 'flat', ' mat']

-- dot star - .\* this literally means it accepts anything anynumber of times, if it encounters a newline it will stop its traversing and returns the string

this is helpful when we have to capture a pattern somewhere in the input and we have no idea what it can have after pattern,it can be clearly explained with an example.

ex:re.compile(r'First name: (.\*) Last name: (.\*) Middle name: (.\*)').findall('First name: badri Last name: pratap Middle name: narayana')

output:[('sai vinil', 'pratap')]

the above example explains well that name can be variable in nature and to capture patterns that vary, we can use .\*

the 'dot star' method is a greedy approach, that is it will try fetching the maximum string that matches the pattern

-- dot star greedy and non greedy approach.

import re

serve='<when you use non greedy>this is also included if you use greedy approach>'

nongreedy=re.compile(r'<(.\*?)>')#to get non greedy approach all we have to do is place a question mark right next to the place in the regex where we want to implement greedy approach.

greedy=re.compile(r'<(.\*)>')

nongreedy.findall(serve),greedy.findall(serve)#the difference between greedy and non greedy is a question mark symbol

--import re

stri='this example will explain \n that a dot star pattern can also accept newline character '

newlineregex=re.compile(r'.\*',re.DOTALL)

newlineregex.findall(stri)

o/p:['this example will explain \n that a dot star pattern can also accept newline character ',

'']

with out re.DOTALL it will it will get a list like this ['this example will explain ','',' that a dot star pattern can also accept newline character ','']

--import re

stri='Agent Bob had handed over the documents to Agent Charlie'

substituteregex=re.compile(r'(Agent) (\w)\w\*')

substituteregex.sub(r'\1 \2\*\*\*\*',stri)

note:the above example illustrates the usage of groups and substituting them with the help of sub function,

the output we get here for re.findall(substituteregex(stri) is [('Agent', 'B'), ('Agent', 'C')], each time when a pattern is found and it checks for replace ment string as we used numbers here it will check for 0th index of the [('Agent', 'B'), ('Agent', 'C')] as zeroth index has ('Agent', 'B') \1->Agent,\2->B , it will search for further pattern and takes element at 1st index i.e, is ('Agent', 'C') here also it replaces \1-> by Agent,\2-> by C which roughly translates the string to

"Agent B\*\*\*\* had handed over the documents to Agent C\*\*\*\*"

--re.verbose #this will help you get a detailed explanation on how to give comment for complex regular expression so that it will be easy for further reference

example:import re

re.compile(r'''#regex for telephone number matching

(((\d\d\d-)| #without paranthesis and with dash note the pipe symbol

(\(\d\d\d\))) #with paranthesis and without dash

\d\d\d #first 3 digits

- #dash

\d\d\d\d #last 4 digits

(\sx\d{2,4})?) #area code optional

''',re.VERBOSE).findall('123-123-1234 123-123-1234 x123 (123)123-1234 x12')

--vowel\_regex=re.compile(r'[aeiou]',re.I)

vowel\_regex.findall('hey VINIL , why cant you maximize your potential when you know you can be much better than you already are')

#here even though we did not mention capital letters in re.compile by passing it will ignore the case and simply match the letters

#you can use re.IGNORECASE inplace of re.I

--in re.compile function , you can use only 2 arguments first one is the pattern and second one is the parameter ex:re.DOTALL,re.I but if you want to pass more than one function as a parameter you have to use bitwise or '|' parameter.

ex:import re

re.compile(r'''#regex for telephone number matching

(((\d\d\d-)| #without paranthesis and with dash

(\(\d\d\d\))) #with paranthesis and without dash

\d\d\d #first 3 digits

- #dash

\d\d\d\d #last 4 digits

(\sx\d{2,4})?) #area code optional

''',re.VERBOSE | re.DOTALL | re.I).findall('''123-123-1234 123-123-1234 X123 (123)123-1234 x12''')

-- ? in regex pattern means the pattern can come zero or one times

###############################################

# zip\_functionality:

when there are multiple lists of equal sizes and you want to group elements by index then you can use zip function

zip([[89.0, 90.0, 78.0, 93.0, 80.0], [90.0, 91.0, 85.0, 88.0, 86.0], [91.0, 92.0, 83.0, 89.0, 90.5]]) will be giving a zip object.

which will have [(89.0, 90.0, 91.0),(90.0, 91.0, 92.0),(78.0, 85.0, 83.0),(93.0, 88.0, 89.0),(80.0, 86.0, 90.5)]

to operate on it:

for i in zip(\*subjects\_marks\_list): #we have to use star as mentioned in the for loop and i stores tuple information,unpack operator is used here

# list\_func:

--if and else in list comprehension

ex:[f(x) if condition else g(x) for x in sequence]

--empty list is treated as false in python **every iterable- list, set, tuple, dictionary are treated as False**

This is useful in cases when you want to check length of the list

--very very important in competitive programming:

to divide a list of size n in to m equal list of lists

li=[1,2,3,4,5,6,7,8,9]

no\_of\_sublists=3

sub\_li=[li[i:i+no\_of\_sublists] for i in range(0,len(li),no\_of\_sublists)] o/p:[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

--to join 2 lists you can follow the above method

l=[]

l.append([1])

l.extend([2])

l#check output

if you want to get the last element of a list, instead of checking for the count and then passing it as a value as l[value], we can just pass l[-1], because the index for first element is 0 and (python counts backwards and in real numbers the number before 0 is -1) so similarly we can use -1 to call/retrieve the last element

'''but the above method will add them to first list, but there might be a scenario where you want to keep the both the lists intact and create a new list which will be obtained by joining the both lists, consider the below scenario'''

l=[1,2,3];l1=[4,5,6]

joined\_list=[\*l,\*l1]#or you can use l+l1

-when you want first input into first list second input into second list and so on...

l=list(range(15,30))

no\_of\_sub\_lists=4

sub\_li=[[] for \_ in range(no\_of\_sub\_lists)]

for i,v in enumerate(l):

sub\_li[i%no\_of\_sub\_lists].append(v)

sub\_li

simplified approach for the above example:

li=[1,2,3,4,5,6,7,8,9]

no\_sub\_li=3

sub\_li=[[] for \_ in range(no\_sub\_li)]

[sub\_li[i%no\_sub\_li].append(v) for i,v in enumerate(li)]

sub\_li

-when you assign a list a to b as

a=[1,2,3]

b=a.copy()

c=a

a[0]='hello'

a,b,c #o/p:(['hello', 2, 3], [1, 2, 3], ['hello', 2, 3])

and if you print a, you will get ['hello',2,3] so it will be a major blunder(this happens because on normal assignment both lists will point to same memory and changes made using one variable will reflect in another as well)THIS NOT ONLY HAPPENS WITH LISTS IT HAPPENS WITH ALL MUTABLE VALUES, if you do assignment directly,so just use .copy() function which will avoid the mistake

a=[1,2,3]

b=a

b[0]='hello'

--in copy module shallow copy method(copy.copy) just copies reference of the object but not the duplicate object, the deep copy method will create a duplicate object for each object iteratively,they make difference only when the iterables are mutable elements check the below example:

# importing "copy" for copy operations

import copy

li1 = [1, 2, [3,5], 4]

li2=copy.copy(li1)

li3=copy.deepcopy(li1)

li2[2][1]=980

li3[2][1]=890

li2[0]='li2'

li3[0]='li3'

print(li1,li2,li3)

#in the above example we changed elements of li2 and li3 at index 0 and 2 but it reflected in li1 at index 2 but not at index 0 as at index 0 it is not a mutable whereas item at index 2 is a mutable element.

--a list on slicing will return a new list

[1,2,3,4,5][1:3] will return [2,3]

--[1,2,3]+[4,5,6]->[1,2,3,4,5,6]

--list({1:2,3:4}) #will return [1,3] i.e, list(dict) will return the keys as list

--difference between list and tuple is that tuple is immutable whereas list is immutable

-- the values in the list are called items

--you can convert a value in to list by passing it in to list(value), but the passed value must be compatible.

-- len(A\_list) will give length of A\_list

l=[1,2,2,4]

l.index(2)

#o/p:1- that will give the position of the FIRST OCCURENCE of 2 in the list

--range(0,8,1) #will return a range object,but to get a list obj use the below code

list(range(0,8))

output:[0, 1, 2, 3, 4, 5, 6, 7]

in addition to above example

- if we dont pass staring index to range object it will behave in abnormal way, for the below syntax it will return an empty list because it will think of 8 as start index and 1 as end index as start index is smaller than end index it will return an empty list.

list(range(8,1))

to get output from this type of operation you have to give negative step as given in the example below:

--list(range(8,2,-1))#o/p:[8, 7, 6, 5, 4, 3]

--rat=[0,1,2]

zero,one,two=rat

zero#o/p:0

--list1=[99,88,77,66]

for i,j in enumerate(list1):

print(i,j)

will store index value in i and item value in j

-- to remove duplicates the list we should use

list1=list(dict.fromkeys(list1))

or we can use

list1=list(set(list1))#this is not preferred as it doesnot maintain order and it wont work for problems where order is specific,mostly it returns the list in sorted order

--a=['a','b','d','c']

a.sort() will save ['a', 'b', 'c', 'd'] in a and a.sort() will return None type but sorts the elements in the list in which it is declared

a.sort(reverse=True)# will return the list in reverse sorted order

''.join(a) will return 'abcd'

to return a list which is sorted you have to use sorted(iterable) which will return sorted list, but the order in the source list will not get sorted

ex: a=['a','b','d','c']

note:items in the list are sorted based on ASCII BETICAL ORDER rather than ALPHA BETICAL ORDER

ex:l=['ant','Aant','Bat','bat']

l.sort()

print(l)#the output is ['Aant', 'Bat', 'ant', 'bat'] because the starting letter of Ant is A and its ascii value is 65 but a is 97

to get them in normal sorting order you have to pass an argument key=str.lower

l=['ant','Aant','Bat','bat']

l.sort(key=str.lower)

print(l)#o/p:['Aant', 'ant', 'Bat', 'bat']

--b=sorted(a)

a will return ['a','b','d','c'] whereas b will return ['a','b','c','d']

b=sorted(a,reverse=True)

b will return ['d','b','c','a'] that is in reverse of sorted order

note the difference between sorted and sort functions sorted will return the sorted list with out changing the parameter that is passed to it, whereas listname.sort() will return None but changes the listname from which the function is called.

--to implement addition of elements position wise of two lists we use

import numpy as np

a1= [1,2,3]

b1= [3,2,1]

a=np.array(a1)

b=np.array(b1)

list(a+b)

--a = [1,5,8]

b = [3,4,7]

result = [min(i) for i in zip(a,b)]--zip will match elementwise and i will store first zipped element of a,b i.e, (1,3),(5,4),(8,7)

result-o/p:[1, 4, 7]

--a1= [1,2,3]

b1= [3,2,1]

c1=[1,1,1]

[sum(i) for i in zip(a1,b1,c1)]

--arr\_list = [1,4,6,8,10,11]

a, \*b, c = arr\_list

a--1

b--[4, 6, 8, 10]

c--11

--[2]\*4 will give [2,2,2,2]

or

[2,]\*4 will give [2,2,2,2]

-l=[2]\*4

l[0]=l[0]+1

l

o/p:[3, 2, 2, 2]

--sorting a list should be done on a list of homogeneous items

--a=[0,1,2,3]

for a[-1] in a:

print(a)

print(a[-1])

output-

[0, 1, 2, 0]

0

[0, 1, 2, 1]

1

[0, 1, 2, 2]

2

[0, 1, 2, 2]

2

--list is a sequence DS,a heterogeneous DS

--list.insert(index,value) will insert value at a given index and pushes remaining elements to the right

--list.remove(value) will remove the first occurence of that value in the list pushing all the elements to the left

--del list\_name[index] will remove the value from the list and reflects it in the list dynamically

the difference between remove and del function is that we use remove when we dont know where the element resides in the list(here we should remember that remove function deletes the first occurence of the element to be deleted) and we use del when we have an element index at our hand which is to be removed and also del is useful when we know that we should delete the element based on its position not on its content.

--a=l.pop() will remove the right most value from list l and assigns it to a

a=l.pop(2) will remove the element at index 2 and assigns it to a

--a=[1,2,11,110]

print([int(i) for i in sorted([i for i in [str(i) for i in a]])])#output:[1, 11, 110, 2]

sorted in lexicographic sorted order for digits, also notice that you have sorted using list comprehension in another comprehension which is a beautiful way of doing it, also you printed directly (in one line) using list comprehension.

--

l='HACK 2'.split()

string=l[0]

count=int(l[1])

[print(i) for i in sorted([''.join(i) for i in list(permutations(string,count))])]

--s,n='HACK 2'.split()

print(\*[''.join(i) for i in permutations(sorted(s),int(n))],sep='\n')# this also serves the same purpose but it uses unpack operator and prints all unpacked values by seperating them by a '\n'

-print('sai','vinil','pratap',sep='space')

#this gives the observation that print function has keywordargurments end and sep

print(sorted(a))#sorted in normal order

-from itertools import combinations

l='HACK 2'.split()

print(\*[''.join(j) for i in range(1,int(l[1])+1) for j in combinations(sorted(l[0]),i,)],sep='\n')#here it is different from the above approach in the range of 1 to n and then the value is passed to combinations.

## Sort list of dictionaries based on value

sorted\_dict = sorted(dict, key = lambda x:x[‘key’],reverse=True)

# dict\_func:

-- a dictionary contains at least one key and value pairs, separated by comma

D = {'key1': 'value1', 'key2': 'value2'..}

If you want to fetch keys alone, use d.keys()

If you want to fetch values alone, use d.values()

If you want to fetch keys and values as a tuple (because you never want these to be muted as tuples are immutable), use d.items()

--dict(a='b',c='d') #this is another way of declaring a dictionary which will be stored as {'a': 'b', 'c': 'd'}

--Dictionaries preserve insertion order. Note that updating a key does not affect the order. Keys added after deletion are inserted at the end. The Dictionary order is guaranteed to be insertion order.

-- to check wether the key is already present in a dictionary or not, you can just use the "in" keyword

1 in {1:2,3:4} # will return true

2 in {1:2,3:4} # will return false

--a={0:1,2:3,4:5}

del a[0]

a#o/p;{2: 3, 4: 5}

a.clear()#will remove all elements but dictionary will remain

del a#will remove whole dictionary elements alongside it's existence

d=a#will point to same memory,i.e,changes made in one dict will reflect in another

d=a.copy()#will point both dictionaries to different memory,i.e, changes made are independant of other dict

--x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}#play with this(incomplete)

{k: v for k, v in sorted(x.items(), key=lambda item: item[1])}

--x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}#play with this(incomplete)

l=sorted(x)

p={}

for i in l:

p[i]=x[i]

print(p)

-- to have multiple values assigned in the from of key to a list, we need to give the below code, but below to this there is a simple approach

-

d={}

d.setdefault(1,[]).append('list1')

d.setdefault(1,[]).append('list1')

d.setdefault(2,[]).append('list2')

#format-dict.setdefault(key,[]).append(value)

print(d)

#o/p:{1: ['list1', 'list1'], 2: ['list2']}

or you can use the below example:

d={}

d[1]=[]

d[1].append('list1')

d

to eliminate the duplicates in the above process you should use set:

d={}

d.setdefault(1,{}).append('list1')

d.setdefault(1,{}).append('list1')

d.setdefault(1,{}).append('list2')

#o/p:{1:{'list1'},2:{'list2}}, check more information on help({}.setdefault)

-from collections import defaultdict

d = defaultdict(list)

d['python']#o/p: defaultdict(list, {'python': []})# as the above parameter is list , by default the items appended will be a list.

d['python'].append("awesome")

d['something-else'].append("not relevant")

d['python'].append("language")

d=dict(d)

d#o/p:{'python': ['awesome', 'language'], 'something-else': ['not relevant']}

--a={0:1,2:3,4:5}

print((a.popitem()))#will return (4,5) the right most item as a tuple

--#dictionary comprehension

d={i-1:i for i in range(1,10)}

d

--dict dont follow order and therefore doesnt support index

d={1:2,3:4}

# d[0] #will raise error

e={3:4,1:2}

d==e #will return true

l1=[1,2]

l2=[2,1]

l1==l2#will return false

--d={1:2,3:4}

d.keys(),d.items(),d.values()

#will print (dict\_keys([1, 3]), dict\_items([(1, 2), (3, 4)]), dict\_values([2, 4]))

-- there will be a scenario where you might want to return a default value for a key if it is not found in the dictionary

ex:if you are going to a picnic and you have to check if you got all the items and if there is any item missing you have the choice of assigning a default value,here in this case 0

d={'apples':10,'tables':2,'chairs':5}

d.get('napkins',0)

'napkins' in d#tha above get function simply returns a default value and doesnot add that key into dictionary

--d={'apples':10,'tables':2,'chairs':5}

d.get('napkins',0)

d.setdefault('napkins',3)#will set napkins as key and value will be 3 and this only works when there is no key named napkin in the dictionary d

--the pprint modules pprint() (pretty print) can print a dictionary value clearly,pprint.pformat will RETURN A STRING VALUE ASSIGNED TO A VARIABLE,DOES NOT PRINT UNLESS USED PRINT FUNCTION.

# iterable\_func:

--all(variable) will return true if all elements in the variable/iterable are true,else returns false,an empty iterable also returns true

all([0])#False

all([1,2,3]) #True

# set\_func

set\_func:

--A set is an unordered collection of elements without duplicate entries.,When printed, iterated or converted into a sequence, its elements will appear in an arbitrary order.

-Basically, sets are used for membership testing and eliminating duplicate entries.

print({2,3,4,5,1,2,3,4,5,9,1,2,6})

o/p:{1, 2, 3, 4, 5, 6, 9}

-set('HackerRank')#o/p:{'H', 'R', 'a', 'c', 'e', 'k', 'n', 'r'}

-set((6,1,2,3,4,5,5))#o/p:{1, 2, 3, 4, 5, 6}

-set(['H','a','c','k','e','r','r','a','n','k']) #o/p:{'H', 'a', 'c', 'e', 'k', 'n', 'r'}

-set({'Hacker' : 'DOSHI', 'Rank' : 616 }) #o/p:['Hacker', 'Rank']

-set(enumerate(['H','a','c','k','e','r','r','a','n','k']))#o/p;{(5, 'r'), (2, 'c'), (8, 'n'), (9, 'k'), (0, 'H'), (6, 'r'), (4, 'e'), (3, 'k'), (1, 'a'), (7, 'a')}

-below are commonly operated methods on set

myset = {1, 2} # Directly assigning values to a set

myset = set() # Initializing a set, you cant assign empty plower braces because that is treated as dict in python

myset = set(['a', 'b']) # Creating a set from a list

myset.add('c')

print(myset) #o/p:{'a', 'c', 'b'}

myset.add('a') # As 'a' already exists in the set, nothing happens

myset.add((5, 4))

print(myset) #o/p:{'a', 'c', 'b', (5, 4)}

myset.update([1, 2, 3, 4]) # update() only works for iterable objects

print(myset) #o/p:{'a', 1, 'c', 'b', 4, 2, (5, 4), 3}

myset.update({1, 7, 8})

print(myset) #o/p:{'a', 1, 'c', 'b', 4, 7, 8, 2, (5, 4), 3}

myset.update({1, 6}, [5, 13])

print(myset) #o/p:{'a', 1, 'c', 'b', 4, 5, 6, 7, 8, 2, (5, 4), 13, 3}

myset.discard(10)

print(myset) #o/p:{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 13, 11, 3}

myset.remove(13)

print(myset) #o/p:{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 11, 3}

--# Frozensets

frozenset: it is a data type in python which is almost like set except that a frozen set values once set can't be changed

syntax: frozenset(iterable)

- the following operations union,intersection,difference and symmetric difference can also be performed on set and the operational logic remains the same way we perform them on normal sets.

# initialize A and B

A = frozenset([1, 2, 3, 4])

B = frozenset([3, 4, 5, 6])

# copying a frozenset

C = A.copy() # Output: frozenset({1, 2, 3, 4})

print(C)

# union

print(A.union(B)) # Output: frozenset({1, 2, 3, 4, 5, 6})

# intersection

print(A.intersection(B)) # Output: frozenset({3, 4})

# difference

print(A.difference(B)) # Output: frozenset({1, 2})

# symmetric\_difference

print(A.symmetric\_difference(B)) # Output: frozenset({1, 2, 5, 6})

--{1,}\*4 will throw an error

-- to read pdf file content the best package is PyPDF2 which reads almost all pdf files and cant read images and other things, should be opened in readbinary mode

import os,PyPDF2

pdfobj=open(path,'rb')

reader=PyPDF2.PdfFileReader(pdfobj)

reader.numpages#gives page count

pageobj=reader.getPage(0) #will return the first page as an object, index starts with 0,i.e, zero index returns page 1

text=pageobj.extractText() #will return the text in string format.

for pagenum in range(reader.numpages):

print(reader.getPage(pagenum).extractText())#will print text in a pdf page by page

#similar to reading a pdf, we can also write page object to a pdf

writer=PyPDF2.PdfFileWriter()

writer.addPage(pageobj)# we can add pageobjects which are obtained from reader.getPage method

#now the content is just in computer memory, now to write it into a pdf file we have to create a new pdf file or open an existing one

output\_file=open(path,'wb')

writer.write(output\_file)#this will save exact contents including symbols images or everything as it is , because we are writing page objects directly.

output\_file.close()

# import:

--import sympy

sympy.isprime(7)

--from datetime import datetime as dt

# from datetime import timedelta

timestamp=(dt.now()).strftime('%Y-%m-%d %H:%M:%S')

timestamp output:'2020-02-05 17:01:01'

from itertools import product

'''if you have two lists of same or different sizes and you want one to one mapping for each element in the list then you can use product'''

a=[[1,2,3],[4,5]]

print(\*product([1,2,3],repeat=2))

print(list(product([1,2,3],[4,5])))

print(list(product(\*a)))#\* used here is called unpack operator in python the unpacking operator simply removes the data type(list,tuple or dictionary) in which the variables are stored and just send the items that are there in the datatype, the below example program gives you a clear idea

def num\_sum(item1,item2,item3):

return item1+item2+item3

num\_sum(\*[1,2,3])

--import smtplib

conn=smtplib.SMTP('smtp.gmail.com',587)#the first parameter is smtp server domain name and the second parameter is 587

'''

gmail.com->smtp.gmail.com, port->587

outlook.com/hotmail.com->smtp-mail.outlook.com ,port->587

yahoo mail ->smtp.mail.yahoo.com ,port->587

AT&T-> smtp.mail.att.net , port ->465

comcast->smtp.comcast.net ,port->587

verizon->smtp.verizon.net, port->465

'''

conn.ehlo()

'''output:

250, b'smtp.gmail.com at your service, [117.195.81.65]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'''

'''the output 250 is the code , code in multiple of hundred and starts with 2 is a successful response, next b'smtp.g... means a binary code'''

conn.starttls()

'''output:

(220, b'2.0.0 Ready to start TLS'), this starts encryption so that you can login

'''

conn.login('pratapsaivinil@gmail.com','ironMan@1')

'''

output:(235,b'2.7.0 Accepted')

as i said above code that starts with 2 is success

'''

conn.sendmail(frommail,tomail,'Subject: enter subject here.. \n\n enter body of the mail\nsecond line\n third line \n\n regards,\nvinil')

'''output:{}

the empty dictionary(i can also call it as a set, but it is a dictionary because if you get any error while executing sendmail method,then it will return an dictionary with key,value pairs that it fail to sent

)'''

conn.quit()

'''output:

(221,b'2.0.0 closing connection........<some text>)'''

-- openpyxl package can be used to create sheets and edit them and save them in a workbook at a path of our requirement

import openpyxl

workbook=openpyxl.load\_workbook(r"D:\genome data\member data scenarios new.xlsx")#here we can use the relative path as well if needed,this will return an existing workbook to read

workbook.get\_sheet\_names() #this will return all the names of the sheets that are present in the workbook

first\_sheet=workbook.get\_sheet\_by\_name('member data scenarios')#this will return a sheet object which will have all the data in form of cell objects

first\_cell=first\_sheet['A1'] #this will return a cell objects and the input A1-> A column,1st row

print(first\_cell.value) #we can also type cast it to the type we want

# first\_sheet.cell(row=0,column=2)#this will through error as row or column value starts with 1

first\_sheet.cell(row=1,column=1)#this is equivalent to first\_sheet['A1'] but this method help us to iterate in a forloop easily.

first\_sheet['A1']='temp'

print(first\_cell.value)

# workbook.save('D:\genome data\member data scenarios new.xlsx')#this will save the changes made on the workbook till now to a specified path,using the same path to save as the original file might not be a good practice as if we make any error in program and save it, we might not be able to retreive it so save the changed file with a new name.

wb=openpyxl.Workbook()#this will create a new workbook object all further operations on this object can be saved to a new excel

wb.get\_sheet\_names()#the newly created sheet object will have only one sheet named 'Sheet'

sheet=wb.get\_sheet\_by\_name('Sheet')

sheet['A1']='first value'

sheet2=wb.create\_sheet()#this will not only create a sheet object but also returns the newly created sheet object and assigns it to sheet2 variable in this code

sheet2.title #will return the name of the sheet object which is currently stored

sheet2.title='second sheet'#this will assign the newly typed name in the form of a variable assignment that is done in general

wb.create\_sheet(index=0,title='sheet at 1st position') #index starting from 0 means the sheet created here will be at first position at excel

# wb.save(r"C:\Users\saivinil.pratap\Desktop\person.xlsx")

--reading emails:

import imapclient

conn=imapclient.IMAPClient('smtp.gmail.com',ssl=True)

conn.login('gmail id','password')

conn.select\_folder('INBOX',readonly=True) # the first parameter here is (in most of the cases will be inbox)

UIDs= conn.search(['since 20-Aug-2015']) #this will return a list of unique id's

for further information refer to automate the boring stuff book

-pyautogui is an useful module helps in automating screen to perform routine tasks easily

import pyautogui

height,width= pyautogui.size() #this will give the screen resolution size

pyautogui.position() # this will give the current cursor position co ordinates, starts at 0,0

# pyautogui.moveTo(100,100,1) #here the first 2 parameters suggest x and y positions to which the cursor is to be moved, if you give x or y values beyond the resolution point, it will raise FailSafeException which denotes the program execution stops, the third parameter is speed(in seconds), i.e, it should take exactly specified amount of seconds to move to that specific points

# pyautogui.moveRel(-10,-80,1) #this is also moving the cursor,but it moves it from the current position, the third parameter is speed in seconds, if you use negative value for x it will move towards left and negative value for y will move it up north

pyautogui.position()

# pyautogui.click(787,232)#the parameters here are optional, if you dont give any paramters it will click at where ever is the cursor before executing the current statement

pyautogui.click(pyautogui.moveRel(10,10,1))#this is also valid

pyautogui.click(350,350,1)#this will take the cursor to the specified poisiton and click at the given position

pyautogui.rightClick(350,350,1)

pyautogui.doubleClick(700,700,1)

pyautogui.middleClick()

pyautogui.leftClick()

# pyautogui.dragRel(x,y)#similar to moveto and moveRel but the cursor is dragged

# pyautogui.dragTo()

# pyautogui.displayMousePosition()# this does not work in idle but works in command prompt and helps us in locating x and y positions of cursor easily and note down it's x and y co ordinates, it also shows the values of the point where cursor is positioned in real time

'''

this module is a great way of automating stuff as it can go to any program it want and click on anything or do anything we can do with a mouse, we can also automate keyboard and we will do that after a while

there might be a scenario where the automated program goes out of control and starts doing something which we doesnt want to do, python automation code for mouse moves it at a speed of 1/10th part of a second as fast as we can we should be moving it to the any corner on the screen which by default will raise a failsafeException which will stop the execution of a program

'''

-- keyboard automation in python

import pyautogui

pyautogui.typewrite('randomtext',interval=0.2) #before you execute this query it is needed to place the cursor in the typable area, giving interval gives human touch and also it is preferable beccause the system may not react as fast as the python does and it might lead to unpredictable results

pyautogui.typewrite(['a','b','left','left','X','Y'])#python lets you perform a series of operations when passed through a list, the current example will type a,b and then press the left arrow twice and then press X,Y

'''to know the keys that are there in the keyboard you can use the below syntax'''

pyautogui.KEYBOARD\_KEYS#this will suppport all the keys that exist such as numlock,home and wide range of variety of keys

pyautogui.press('volumeup')#will press the key,here it will increase the volume

pyautogui.hotkey('ctrl','o')#there will be scenarios where you might want to press more than 1 keys at a time then you can use this hotkey method

below is a sample code of automation to open a notepad and write some info into it

-

import pyautogui

import time

pyautogui.click(pyautogui.moveTo(340,1040,1))

pyautogui.typewrite('notepad++',interval=0.2)

pyautogui.doubleClick(pyautogui.moveTo(670,400,1))

pyautogui.press('escape')

time.sleep(1)

# pyautogui.typewrite(['alt','f'])

pyautogui.hotkey('alt','f')

pyautogui.press('N')

pyautogui.typewrite('i did this purely using python code but it took me around 2 hours lol',interval=0.2)

--taking a screenshot and searching

'''in the above automations of keyboard and mouse it will blindly try to do whatever we have coded, if we try to type on a

video player that will not work, so to give eyes to the program we can use screenshot featurein pyautogui which we will discuss in detail now

you can take screenshot normally if you install pyautogui if you are and if you are a linux used you have to install scrot by running command sudo apt-get install scrot

'''

a=pyautogui.screenshot() # this will take screenshot and will store it in a variable, but to actually save it in a file, we need to send path as a parameter

pyautogui.screenshot(r"C:\Users\vinil\OneDrive\Desktop\screenshot.png")

pyautogui.locateOnScreen(r"C:\Users\vinil\OneDrive\Desktop\cal7-4.png")

'''in the above syntax we already saved a particular image which will be used to find wether

there is a match that is there in the current screen in the above example i cropped the window

logo that will be there on the bottom left and searched for it using the above syntax passing

the cropped image as parameter, it will return a tuple with its x,y positions and width and height

of the logo, the drawback here is it will try to match the image pixel wise i.e, if you enlarge it

os minimize it it will not return the expected result(i presume)'''

pyautogui.locateCenterOnScreen(r"C:\Users\vinil\OneDrive\Desktop\cal7-3.png")#this works exactly as above syntax,but it will not return the hight and widtgh of the matching image, instead it will locate the center of it so ,it can have more accuracy of clicking the target

-- python allows creation and editing of docx files

import docx

d= docx.Document(path)# this will create a document object with the contents of the document that is passed as a path,let it be a doc or docx file.

'''the d object used above is having multiple paragrapgh objects provided that the document has multiple paragraphs.the paragraph objects are returned as lists'''

paras\_list=d.paragraphs

paras\_list[0].text# this will return the text that is in the first paragraph

'''a paragraph in a document can have many types of text varying at style or type of the font, below method returns the runs which will divide the para into partitions based on the type of the text i,e. suppose we have a text which is having normal words and then bold and then normal text, it is divided in to 3 runs, the first normal words are grouped into one run, then the bold text is grouped in to one run,then the normal text is categorised into another run.'''

runs\_list=paras\_list[0].runs

'''after this we can also get the type of the text it is holding,'''

runs\_list[1].text #will return the bold as it is bold type

runs\_list[1].text='this text is replaced'#you can even assign a new text like this and update the document

runs\_list[1].underline=True

'''as shown above the text member variable will deal with bold italic underline, to deal with styles we have to use style method.'''

paras\_list[0].style='Title'

'''you can also create a new document and add text and changes styles just like you do with opening an existing docx file'''

new\_doc=docx.Document()# this will create a document and this document only exist in python memory and needs to be saved if you want to use it for further referrence

new\_doc.add\_paragraph('this is adding a text to paragraph)

new\_doc.save(path)#this will save the newly created document in the specified path, let it be a doc or docx file

p=new\_doc.paragraphs[0]

p.add\_run('this will add new text at the end of para 1')

p.runs[1].bold=True #this will make the above newly inserted text as bold

d.save(path) #will store the docx file with all changes reflected

-'''below sample program will return the text of the any document.'''

def return\_text(doc):

para\_list=doc.paragraphs

stri=''

for i in para\_list:

stri+=i.text+'\n'

return stri

doc=docx.Document(r"C:\Users\vinil\Downloads\demo.docx")

print(return\_text(doc))

--import mysql.connector.connect#is used to retrieve data from database

db=mysql.connector.connect(host='hostname',database='db\_name',user='username',password='password')#connection is stored in db variable

df=pd.read\_sql(sql\_query,con=db)#connection\_stored\_variable=db

--import statsmodels.datasets#it contains real world data analysis

data=statsmodels.datasets

-- see import textblob,contractions#here you have to observe that we have imported to modules by seperating them with a comma

--import random

print(random.randint(1,10))

is same as

from random import \*

print(randint(1,10))

-import pyperclip

pyperclip('some random text')#the text in the brackets is copied to clipboard which can be used after exiting from python and used to paste using ctrl+v ,this syntax can be used when you have to export some large amount of text outside script without saving it into any external file andklk to keep it on temporary cache

pyperclip.paste()#this will take some text which is stored in clipboard of ram and bring it to the variable to which it is assigned.

## Absolute and relative imports

### What happens when you import a package or module in python

## Json

--The full-form of JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called json. To use this feature, we import the json package in Python script which provides us with a lot of methods which among loads()(loads string as a json object) and load() (loads json file as json object)methods are gonna help us to read the JSON file. The text in JSON is done through quoted string which contains the value in key-value mapping within { } similar to dictionary.

ex:

import json

data = {

"name": "Satyam kumar",

"place": "patna",

}

datas = '''{

"name": "Satyam kumar",

"place": "patna"

}'''

with open(r"C:\Users\saivinil.pratap\Desktop\TEMP.txt","w" ) as write:

json.dump( data , write )

with open(r"C:\Users\saivinil.pratap\Desktop\TEMP.txt", "r") as read\_content:

print(json.load(read\_content))#first we created a json file and then loading the content of it by loading a json object using json.load method

print(json.loads(datas))#here we use loads method to load a string object, the string object we load should be a documented string

the following objects in json is read as corresponding elements in python.

object ->Dict, array ->List, string ->Str, null ->None , number (int)->int , number(real) ->float, true ->True, false ->False

load is used when we are loading a json file object,loads is used when we are using json string object

ex1(string obj):

# JSON string

a = '{"name": "Bob", "languages": "English"}'

# deserializes into dict

# and returns dict.

y = json.loads(a)

ex2(file object):

f = open('data.json',)

# returns JSON object as a dictionary

data = json.load(f)

# Iterating through the json

# list

for i in data['emp\_details']:

print(i)

### printing json string with indentation

the below piece of code need a dictionary which is a string, and will print json in neat and indented format

import json

st = '{"captureData":{"endpointInput":{"observedContentType":"text/csv","mode":"INPUT","data":"1294,1,female,22.0,0,1,59.4,C","encoding":"CSV"},"endpointOutput":{"observedContentType":"text/csv; charset=utf-8","mode":"OUTPUT","data":"1","encoding":"CSV"}},"eventMetadata":{"eventId":"4ce0063e-d634-4043-b060-fdbbec91714e","inferenceTime":"2022-09-21T09:59:49Z"},"eventVersion":"0"}'

print(json.dumps(json.loads(st), indent=4))

# Tuple

--tuples will accept all types of datatypes in python, it is faster than list

--if a function is returning more than one values then the values are returned in the form of tuple.

--l=('sai') is a string not tuple

--a='sai', is tuple,not an error

--a=('sai',) is tuple not an error

--t=(1,[2,3])

print(id(t))

t[1][0]='sai'

print(id(t)) even though tuple is changed it points to the same address(here tuple is edited, it can't be)

--a=('sai',)\*4 output:('sai', 'sai', 'sai', 'sai')

--print(divmod(9,2)) will return a tuple with quotient as first value and reminder as second value

--tuple unpacking:

Splitting tuple elements into individual variables

# Df

dataframe\_func:

--data.sort\_values("First Name", inplace = True) will sort dataframe w.r.t "first name" column and changes are replaced in the column

--data.drop\_duplicates(subset=,keep=,inplace=)#subset will have the columns which will have the duplicates to be removed by default it will be none,keep=first will keep the first unique occurence,keep=last will keep the last occurence of the duplicated record,keep=false will remove all records which are duplicated atleaast once

--df.index #will get the series of the index and it can be converted to list by list(df.index) or get length of it by len(df.index)

--df.index.get\_loc(number)#will return the value stored in the number position.

--df.set\_index(<column\_name>,inplace=True/False)#will make the new given column as index

--df=df.replace(np.nan,'',regex=True)#will replace all the null values with empty strings in a dataframe

--df.drop([list of column names],axis=1,inplace=true)#will remove the columns declared in the list

df1=df.drop([list of column names],axis=1,inplace=false)#df1 will have the df with list of columns dropped but df will have even the dropped columns

--df[['SBSB\_ID', 'MEME\_CK', 'SBSB\_CK', 'GRGR\_CK', 'APAP\_ID']].head(5)-#will give the top 5 values of all those dataset columns that are mentioned here

--df.shape#will give the rowXcolumn information

--df.columns#will give the column information of the dataframe df

--df.describe(percentiles=[0.2,0.4,0.6,0.8],include=['object','int','float'])

--df.describe() #will give initial statistics of dataframe df,will give the values of columns such as min,max,count,sum,standard deviation and etc values.

rslt\_df = dataframe.loc[dataframe['Percentage'] > 80]

->rslt\_df = dataframe.loc[dataframe['Percentage'] != 95]

->rslt\_df = dataframe.loc[dataframe['Percentage'] != 95]

-># selecting rows based on condition

rslt\_df = dataframe[dataframe['Stream'].isin(options)]

-># selecting rows based on condition

rslt\_df = dataframe.loc[dataframe['Stream'].isin(options)]

-># selecting rows based on condition

rslt\_df = dataframe.loc[~dataframe['Stream'].isin(options)]

-># selecting rows based on condition

rslt\_df = dataframe[(dataframe['Age'] == 21) &

dataframe['Stream'].isin(options)]

-># selecting rows based on condition

rslt\_df = dataframe.loc[(dataframe['Age'] == 21) &

dataframe['Stream'].isin(options)]

-># sort by index labels

sample\_df.sort\_index(axis = 0)

--# sorting based on column labels

df.sort\_index(axis = 1)

--sorting a dataframe based on index(when axis value is not declared it is zero and sorts rows) and inplace=true will reflect changes in dataframe

df.sort\_index(inplace=True)

--set\_index method in dataframe:

df.set\_index('month')

year sale

month

1 2012 55

4 2014 40

7 2013 84

10 2014 31

Create a MultiIndex using columns ‘year’ and ‘month’:

>>> df.set\_index(['year', 'month'])

sale

year month

2012 1 55

2014 4 40

2013 7 84

2014 10 31

Create a MultiIndex using an Index and a column:

>>> df.set\_index([pd.Index([1, 2, 3, 4]), 'year'])

month sale

year

1 2012 1 55

2 2014 4 40

3 2013 7 84

4 2014 10 31

Create a MultiIndex using two Series:

>>> s = pd.Series([1, 2, 3, 4])

>>> df.set\_index([s, s\*\*2])

month year sale

1 1 1 2012 55

2 4 4 2014 40

3 9 7 2013 84

4 16 10 2014 31

--DataFrame.set\_index(self, keys, drop=True, append=False, inplace=False, verify\_integrity=False):\

drop : bool, default True

Delete columns to be used as the new index.

append : bool, default False

Whether to append columns to existing index.

inplace : bool, default False

Modify the DataFrame in place (do not create a new object).

data.set\_index(["First Name", "Gender"], inplace = True,

append = True, drop = False)

df.set\_index('pd.Index([5,6,7,8])')#length should be equal to number of rows

data.reset\_index(inplace=True)# this resets the previous index

data['DocName']=data.index

--import string;set(string.punctuation)

--punc\_lgc = str.maketrans('ab','cd') #punc\_lgc is a string

punctuationremoval = document.translate(punc\_lgc) #after this line in punc\_lgc 'a' will be replaced by 'c' and b will be replaced by 'd'

-- df.shape[0] #which will always correctly tell you the number of rows (dont use df.count() will return no of NaN values), because df.shape will return a tuple with two values, (row\_count,col\_count) df.shape[0] will return row count

--df.append([tempdf[0:int(0.2\*tempdf.shape[0])]['Word'].values])-will append 20 values of 'word' column to df dataframe

--df= df[df['APAP\_CUR\_STS']=='CL']

--df2= df1[df['APAP\_CUR\_STS']=='CL']#here df['APAP\_CUR\_STS']=='CL' will return a series which will give the true,false values based on condition,df1 should have equal row length as df1 and now df1 rows will be returned based on true and false values that are returned in the above generated series and all rows are assigned to df2

df2.head(5)

--df['column1'].value\_counts() # will give each value and its corresponding repititive count in 'column1'

--df['column1'].unique() #will return a numpy ndarray which will return unique values in column1 which can be converted to list

--print(df.columns) and print(df.columns.values) will return the columns of that dataframe and the return type of both functions is not same

--#standardize the data to normal distribution

from sklearn import preprocessing

dataset1\_standardized(this is a excel which is converted to dataframe) = preprocessing.scale(dataset1)

--#merging 2 similar dataframes (same columns)

dataf = pd.DataFrame({'Date':['10/2/2011', '12/2/2011', '13/2/2011', '14/2/2011'],

'Event':['Music', 'Poetry', 'Theatre', 'Comedy'],

'Cost':[10000, 5000, 15000, 2000]})

v=dataf.head(2)

v1=pd.concat([v,dataf])

v1

--df.info()#will give all datatypes of columns in df

converts dataframe to csv

df\_bi.to\_csv(r'C:\Users\saivinil.pratap\Desktop\bigrams and unigrams of pdf3.csv')

--print(df.isna()) #will return all columns with missing values in df

--print(df.isna().sum()) #will return all columns and their corresponding null count

# Pytest

## What is pytest

A framework which makes building simple and scalable (i.e., scales to support complex functions supported by applications and libraries) tests easily

## Advantages of using pytest

* open source
* can skip tests
* can detect tests automatically
* run tests in parallel
* can run specific tests or subset of tests
* easy syntax and simple to start with

## rules of pytest

## should we be using py.test or pytest while executing pytest commands

we can use either

### to detect test files automatically

for pytest to detect folders or files or methods **automatically**, they should have a prefix of **test\_**

ex: **test\_**folder/**test\_**file.py

in **test\_**file.py you need to have methods such as

def **test\_**sum:

pass

**pytest will detect files automatically from its directories subdirectories, files and methods under current directories (if you want to run tests inside a particular folder, you can just navigate to that folder in terminal and run “pytest” command) and to run them just execute the command *pytest*** **in terminal, pytest will search** its children folders or children files or methods with test\_prefix and run them for you

## importance of assert in a test case

assert syntax:

assert <expression or value> = <expression or value>

assert is used to check whether expected test value is matching with actual test value

**you can have multiple assert statements in a test case (method), if one of it fails, pytest will move to next testcase ignoring the remaining statements in the current testcase**

## how to run tests

### points to note before running pytest

if you are testing APIs of django, you need to have server running before you run the tests

### running test names by substring matching

we can use this substring mechanism when you have some similar tests with a common substring, pytest will fetch the tests having the passed substring as a part of the function definition

**command: pytest -k <substring> -v**

parameter -k is used for substring match

parameter -v increases verbosity

#### example code

import pytest

def add\_6\_to\_num(num):

    return num+6

def add\_7\_to\_num(num):

    return num+7

def test\_add\_6\_to\_num():

    assert add\_6\_to\_num(3) ==9

    print('hi')

    # assert add\_6\_to\_num(3) ==8

    print('hi1')

def test\_add\_7\_to\_num():

    assert add\_7\_to\_num(3) ==10

    print('hi')

    # assert add\_6\_to\_num(3) ==8

    print('hi1')

commands to run

pytest -k test\_add -v

### you can run all tests in file by running the test file

command used: pytest <location of the test file>

ex: pytest test\_sample.py

### running tests by grouping them in to markers

you can add annotation @pytest.mark.<custom name you want to give to marker>

command to run markers

pytest -m <custom name you want to give to marker>

## fixtures in pytest

fixture definition:

it is a piece of software or a device that sets a system to satisfy certain pre conditions for a software

examples of common fixtures: loading test set to the database, reading a configuration file, setting up environment variables

whenever we need to run some code (running code might be needed because we need to generate some data before running the test case) before running test cases, the code that we run before running the testcase is called fixtures

you have to use @pytest.fixture annotation before the method and that method is converted to a fixture

**below we use fixture function numbers as a parameter to test methods**

import pytest

@pytest.fixture

def numbers():

    return [10,20,25]

def test\_method1(numbers):

    x=15

    assert numbers[0]==15

def test\_method2(numbers):

    x=15

    assert numbers[1]==20

def test\_method3(numbers):

    x=15

    assert numbers[2]==25

**command to run**

pytest <filename.py>

### need for scope in fixtures

when you are using the same fixture for multiple test methods, the fixture method is loaded each time, but if you know that same data is going to be used for all the **functions or class or module or package or a session** you can pass that as a parameter and save the time and execution cost

#### types of scopes

##### function

This scope works well if the fixture is only used once or it contains a very light operation or you want a different value each time.

##### Class

The scope class runs the fixture per test class. If you have a couple of test functions that do similar things, such as arithmetic operations or database queries, you can put them in the same test class with the decorator @pytest.mark.usefixtures("fixture-name"). This special decorator adds the fixture to a test class, and the fixture will be executed before any test function. Check out the logs below.

@pytest.fixture(scope="class")

def dummy\_data(request):

    request.cls.num1 = 10

    request.cls.num2 = 20

    logging.info("Execute fixture")

@pytest.mark.usefixtures("dummy\_data")

class TestCalculatorClass:

    def test\_distance(self):

        logging.info("Test distance function")

        assert distance(self.num1, self.num2) == 10

    def test\_sum\_of\_square(self):

        logging.info("Test sum of square function")

        assert sum\_of\_square(self.num1, self.num2) == 500

# test/test\_code.py::TestCalculatorClass::test\_distance

# ------- live log setup -------

# INFO     root:test\_code.py:59 Execute fixture

# ------- live log call -------

# INFO     root:test\_code.py:65 Test distance function

# PASSED                                                                                                                                         [ 50%]

# test/test\_code.py::TestCalculatorClass::test\_sum\_of\_square

# ------- live log call -------

# INFO     root:test\_code.py:69 Test sum of square function

# PASSED

# source code

def distance(num1, num2):

    return abs(num1 - num2)

def sum\_of\_square(num1, num2):

    return num1 \*\* 2 + num2 \*\* 2

##### module

the fixture is run only once per module

##### package

the fixture is run only once per package

##### session

The last and the most aboard scope is session. Fixtures with scope session will only be executed once per session. Every time you run pytest, it’s considered to be one session. Scope session is designed for expensive operations like truncating table and loading a test set to the database.

### use of yield in pytest

There is a special usage of yield statement in Pytest that allows you to execute the fixture after all the test functions. The code before yield serves as the setup code, and the code after yield serves as the teardown code. A classic example would be a testing database.

@pytest.fixture(scope="class")

def prepare\_db(request):

    # pseudo code

    connection = db.create\_connection()

    request.cls.connection = connection

    yield

    connection = db.close()

@pytest.mark.usefixtures("prepare\_db")

class TestDBClass:

    def test\_query1(self):

        assert self.connection.execute("..") == "..."

    def test\_query2(self):

        assert self.connection.execute("..") == "..."

## parametrize test

we can parametrize the test to run multiple arguments for a test i.e., to pass a list of different set of inputs to the test case

in the below code you have x,y,z values and we are passing two sets of values in the form of tuple of lists

import pytest

@pytest.fixture

def numbers():

    return [10,20,25]

def test\_method1(numbers):

    x=15

    assert numbers[0]==15

def test\_method2(numbers):

    x=15

    assert numbers[1]==20

def test\_method3(numbers):

    x=15

    assert numbers[2]==25

command to run:

pytest <path of the file>

## skipped and xfailed

@pytest.mark.xfail -> will treat the testcase as a xfail instead of fail, and xpass instead of pass

@pytest.mark.skip -> pytest will skip this test

## Testing an API using pytest

# Threading in python

## What is threading in programming

Earlier any task we wanted to be done through programming is converted in to an instruction or set of instructions and those instructions are sent to processor and processor used to execute them one by one. But after a few years, the processor is divided in to parts called cores which behaves like an individual processor and process instructions and gives output, and each core is further divided in to two threads (or more), so that each core can have multiple parts of it (as threads) and also can be given priority and threads with higher priority are given , such that important tasks can be directed to threads with higher priority

A high priority thread is called foreground thread, a low priority thread is called background thread.

A background thread and foreground thread cannot communicate with each other

A high priority thread or a foreground thread has easy access to cpu or processor time compared to low priority thread, a high priority thread has nothing to wait for and can be executed immediately if there are no threads with equal or high priority (than current thread) in execution

**A high priority task which is usually assigned to a high priority thread, cannot be assigned to a low priority thread, if a low priority thread tries to tackle a low priority, then that results in a crash, so it is important to know which thread (high or low priority) you are on and which task (high or low priority) to avoid crashes**

### Analogy on thread priority

# requests module:

link-https://www.geeksforgeeks.org/python-requests-tutorial/

--import requests# a third party module for downloading web pages and files,this will help us import the requests module which can be used to get the data that is diplayed on a webpage using an url

res=requests.get(r'https://en.wikipedia.org/wiki/Wikipedia') # this will return a requests object and store it in res

res.status\_code#this will return the status code of the res object 200-> retrieval is successfull,404->not found, any code that starts with 2 and is of length 3 will be a good response

res.text #this will return the text in raw format which means it is sent from the server'

res.raise\_for\_status()#this will raise an exception if the request does not go well,returns None if the request fetches the data correctly

samp\_file=open('romeoandjuliet.txt','wb')#a file is opened in write binary mode as the content fetched is in binary form

for chunk in res.iter\_content(10000):#iter\_content is used ro get chunks similar to iteritems for dictionary

print(chunk)#each chunk will have 10000 bytes of text

samp\_file.close()

--a simple parse to get an price of the item using beautiful soup and requests module

import bs4,requests

def get\_price(flipkart\_url):

res=requests.get(flipkart\_url) #returns a requests object

res.raise\_for\_status() #this raises an exception if the request object res is not retrieved properly and notice the open and close brackets

soup\_obj=bs4.BeautifulSoup(res.text,'html.parser')#res.text not res , html.parser is required to inform interpreter that the object is of html, not a mandatory one but raises an exception if not mentioned

elems = soup\_obj.select('#container > div > div.\_2c7YLP.UtUXW0.\_6t1WkM.\_3HqJxg > div.\_1YokD2.\_2GoDe3 > div.\_1YokD2.\_3Mn1Gg.col-8-12 > div:nth-child(2) > div > div.dyC4hf > div.CEmiEU > div > div.\_30jeq3.\_16Jk6d') #to get an parameter to be passed to the select object we need to right click on the object we need to parse and then click copy selector in edge ,chrome browsers and it might be mentioned as css in some other browsers

print(elems[0])

return elems[0].text.strip() #the above elems is a list and each element will have a text object, elem object will have tags as well and to remove them we choose elems.text to retrieve the text

price=get\_price('https://www.flipkart.com/realme-narzo-20-glory-sliver-64-gb/p/itm4ac58d879006d')

price

--automating a web browser for some tasks in python

from selenium import webdriver

browser=webdriver.Firefox()#this runs and opens new firefox window

browser.get(url) #this will open a new url in the opened tab of firewfox

elem=browser.find\_element\_by\_css\_selector('<unique css selector>')#for the element to operate on you should have to right click on the element and copy the unique css selector

elem1=browser.find\_elements\_by\_css\_selector('p')# this is similar to above syntax but it will fetch all the elements with p(paragraph) tag

#similarly you can find elements by class,id,link\_text,partial\_link,name,tag\_name

elem.click()#this will click the selected element

#to give an input to a field, you get the unique css selector and generate an element ibject as mentioned above, then follow below syntax

elem.send\_keys('<text to be entered>')

#there is no rule that each input box should have a submit button , but most of them will have , in the above example after entering text you want to click on search, but for that therre is no need to find the search button selector we can do that using elem.submit()

elem.submit() #this also works for submit buttons i think, have to play and search

browser.back() #this will navigate to previous step or previous url opened

browser.forward() # this will navigate forward

browser.refresh() #this will refresh the page

browser.quit() # this will close the browser

#to get a particular text element such as para

elem=browser.find\_element\_by\_css\_selector('<selector id>')

elem.text #this will have the text inside it

#to get all the content pass html or body tag

elem=browser.find\_element\_by\_id('html')

elem.text

--request module have get,post,put,delete,head,patch methods

--get method : will get url as input and the response method (calling request.get/put/post/delete/patch will return a response method) will contain the head and body of the url(text and all other content), that response method is a powerfull object which can give other details about the request method which had given the response object(ex:response.url will return the url which is passed as a parameter in the earlier request method)

--post request : a post request requests a web server to accept data enclosed in the body of the request message , most likely for storing it(url is not given as a parameter here)

--put method : the put method requests that enclosed entity to be stored under the supplied url, if the url is an existing resource then it is modified,or else a new resource is created.

--delete method: deletes the specified resource

--head method : similar to get request it accepts the url and returns the head of the response without body

--patch method : it has modifying capabilities , patch request should only contain the data that needs to be changed in the resource

as requests.method will return a response object there are many operations which we can perform with a response object.

ex:rm = requests.get(url)#here rm is a response object

rm.headers#returns a dictionary of response headers

rm.encoding#returns the encoding used to decode the response content as encoding format differs over a wide range

rm.elapsed#as the name suggests it will return the time elapsed between request sent and response recieved

rm.close()#will close the connection to the server

rm.content#will return the content of the response in bytes

rm.cookies#will return the cookie jar with cookies sent back from the server

rm.history#will return the llist of response objects the holding the history of request

rm.

# OS

## To get all the environment variables inside the python file

Import os

os.environ #will have an environ object which can be type casted to dictionary

var\_dict = dict(os.environ)

#var\_dict stores the list of variablenames as keys and their corresponding values as variables

## To fetch a particular variable

Import os

str\_value = os.environ['DJANGO\_SETTINGS\_MODULE']) #returns a str object

## deleting already existing variable

import os

del os.environ[<var>]

## to not to throw an error while deleting non existent variable

os.environ.pop(<var>, None)

# poetry

poetry is a dependency manager in python.

You can install poetry using conda or pip (pip virtual env is suggested in documentation)

 It allows you to declare the libraries your project depends on and it will manage (install/update) them for you. Poetry offers a lockfile to ensure repeatable installs, and can build your project for distribution (so that it can be shared along peers for installing it in their machine).

## pyproject.toml

this file has some of the meta data and dependencies information which you get after add a new project using poetry or initialize poetry for an existing project

## Poetry commands

### adding poetry to a new project (project that is getting created now from scratch)

poetry new poetry-demo

this will create directory with the following content

poetry-demo

├── pyproject.toml

├── README.md

├── poetry\_demo

│ └── \_\_init\_\_.py

└── tests

└── \_\_init\_\_.py

### poetry init

if you want to initialize poetry for an existing project, this will create a pyproject.toml at the root folder

### poetry install

after you do **poetry init**, you can see that a pyproject.toml got created at the root folder, and you can add/remove the dependencies using **poetry add** or **poetry remove** commands, then if you are doing the poetry install for the first time, then after installation, you can see poetry.lock file getting generated at root folder.

#### poetry install --noroot

If you want to install the dependencies only, run the install command with the --no-root flag:

### poetry lock

if you are manually updating (ex: adding or removing a dependency from) pyproject.toml file you need to update poetry.lock file by running **poetry lock** command.

By default at first **poetry.lock** file is not generated, after you do **poetry install**

#### Important points to note - If poetry.lock already exists

If there is already a poetry.lock file as well as a pyproject.toml file when you run poetry install, it means either you ran the install command before, or someone else on the project ran the install command and committed the poetry.lock file to the project (which is good).

Either way, running install when a poetry.lock file is present resolves and installs all dependencies that you listed in pyproject.toml, but Poetry uses the exact versions listed in poetry.lock to ensure that the package versions are consistent for everyone working on your project. As a result you will have all dependencies requested by your pyproject.toml file, but they may not all be at the very latest available versions (some dependencies listed in the poetry.lock file may have released newer versions since the file was created). This is by design, it ensures that your project does not break because of unexpected changes in dependencies.

### poetry --version

Once poetry is installed in your machine, and its corresponding binary file location is added

### poetry self update

this will update poetry

### poetry update

if you manually update versions in pyproject.toml file, then you need to run command **poetry update,** this will update your packages and poetry.lock file as well

This will fetch the latest matching versions (according to your pyproject.toml file) and update the lock file with the new versions. (This is equivalent to deleting the poetry.lock file and running install again.)

### when you want to install/upgrade to a specific version

poetry self update <version>

### poetry add <package> - this is the way to specify dependencies

this will add the above mentioned package to pyproject.toml file directly and also install that particular version to your python environment

#### adding a version >=

ex: **poetry add bs4^2.1** will install bs4 version which is greater than or equal to 2.1 or equal to 4.12.2 (highest version as of today) and also add it as a dependency in pyproject.toml

#### adding a particular version

poetry add beautifulsoup4=4.12.1

#### adding particular version in range

poetry add plotly">=5.12,<5.13"

### poetry remove <package>

this will remove the package from the environment, also it will remove from the pyproject.toml

# pdb

pdb is a python package which is a debugger that is used to stop the execution at a certain point, where you think that the code might be having a bug

to debug along with the code and also to access the values that the interpreter holds in the previously executed lines, you can use

import pdb

pdb.set\_trace()

or

breakpoint()#no need of any imports for calling breakpoint()

when interpreter encounters set\_trace or breakpoint, it will stop the execution and shows the next line before which it has stopped execution the interpreter

a=1

breakpoint()

print(2)

in the above case, interpreter prints 1 and stays near print(2) without execution, as something like this

-> print(2)

(Pdb)

(Pdb) in the above piece of code shows that the pdb package is active

Arrow pointing to a line states previous lines (to the current line ex:a=1) have been executed and the interpreter is waiting for your operations like you can print the variable ‘**a’**

## Tips

* While debugging, if you are not sure, from where we are getting the exception, then place the whole code in try block and put an breakpoint in exception
* Also to print a stack trace of from where the code is exactly breaking, then give logger.exception(exc\_info =True)

from structlog import get\_logger

logged = get\_logger(\_\_name\_\_)

except Exception as error:

            breakpoint()

            logged.exception(exc\_info=True)

            print(error)

* If you are having nested for loops and a lot of code and you want to place a breakpoint at particular iteration, then

for data in scenario\_offers\_serializers.data:

if scenario\_offers\_serializers.data.index(data) ==6:

breakpoint()

* a

## help –

to get the list of all commands you can type help

-> with uow as unit\_of\_work:

(Pdb) help

Documented commands (type help <topic>):

========================================

EOF    c          d        h         list      q        rv       undisplay

a      cl         debug    help      ll        quit     s        unt

alias  clear      disable  ignore    longlist  r        source   until

args   commands   display  interact  n         restart  step     up

b      condition  down     j         next      return   tbreak   w

break  cont       enable   jump      p         retval   u        whatis

bt     continue   exit     l         pp        run      unalias  where

Miscellaneous help topics:

==========================

exec  pdb

### printing variables and values held by interpreter

let’s say from one python file we are calling a method which is in another python file, but before we call, there might be some variables which are assigned in the parent method, to get the variables that are assigned in parent methods and variables till the current line are stored and will be printed if you type **a** and press enter

if you want to know what does the option a do, you can use **help a**

(Pdb) help a

a(rgs)

        Print the argument list of the current function.

We now know that a will give the list of arguments that function holds, typing a will show the list of variable names and their corresponding values

(Pdb) a

scenario\_ids = '1001,1002'

kpi\_code = '1007'

category = 'segments'

uow = <apps.scenario\_comparision.unit\_of\_work.ScenarioMetricsUnitOfWork object at 0x7f69fe85f220>

### printing a particular variable

if you know a variable has been assigned in before lines, then by typing the name of the variable and pressing enter will give the value of the variable, this can be any variable which is in the current/parent methods

(Pdb) segments\_value

'1001'

#### Imp note

If you want to print a variable a, that will coincide with the inbuilt command a(rgs) so you can use

p (print) or pp (pretty print)

(Pdb) p a

1.0

(Pdb) pp a

1.0

### Interact

By typing this we can open an interactive terminal where we can access global and local names, also execute statements whose code exceeds more than one line

(Pdb) help interact

interact

Start an interactive interpreter whose global namespace

contains all the (global and local) names found in the current scope.

To get out of interactive mode and to get continue with debugging press ctrl+d

### continue or c

typing c will move your code to the next breakpoint, executing all intermediate lines in between

(Pdb) help c

c(ont(inue))

Continue execution, only stop when a breakpoint is encountered.

### list or l

will print few lines above and below the line which is to be executed (the line with the arrow mark ->)

(Pdb) l

         query\_string, sql\_param\_nos = qr.sp\_comparision(category)

         query\_string = query\_string.replace(

             "(%s)", "(" + ",".join(["%s"] \* len(scenario\_ids.split(","))) + ")"

         )

         breakpoint()

  ->     with uow as unit\_of\_work:

             df\_metrics\_raw = unit\_of\_work.get\_data\_df(

                 query\_string,

                 sql\_params[0 : sql\_param\_nos - 1 + len(scenario\_ids.split(","))],

             )

         if df\_metrics\_raw.shape[0] == 0:

### ll (long list)

will print more lines above and below the line which is to be executed (the line with the arrow mark ->)

(Pdb) ll

    def get\_metrics(

        scenario\_ids: str,

        kpi\_code: str,

        category: str,

        uow: unit\_of\_work.DjangoUnitOfWork,

    ):

        segments\_value = (

            StatusEnum.Segments.value

            if category == "segments"

            else StatusEnum.ProductCategory.value

        )

        """Return the metrics based on category, scenario Ids and KPI value input

        Based on the category scenario Ids and KPI, data is then filtered

        from the DB based on the metric names.

        Parameters

        ----------

        scenario\_ids

            List of scenario Ids

        kpi\_code

            Indicates the kpi code

        category

            To indicate the type of category like segments or product or overall.

        uow

            Unit of work to get the data from the DB.

        Returns

        -------

        list

            list of dicts if successful, empty list otherwise.

        """

        sql\_params = scenario\_ids.split(",") + [segments\_value, kpi\_code]

        query\_string, sql\_param\_nos = qr.sp\_comparision(category)

        query\_string = query\_string.replace(

            "(%s)", "(" + ",".join(["%s"] \* len(scenario\_ids.split(","))) + ")"

        )

        breakpoint()

 ->     with uow as unit\_of\_work:

            df\_metrics\_raw = unit\_of\_work.get\_data\_df(

                query\_string,

                sql\_params[0 : sql\_param\_nos - 1 + len(scenario\_ids.split(","))],

            )

        if df\_metrics\_raw.shape[0] == 0:

            raise NoDataError(sql\_params)

        # handle application specific exceptions

        df\_metrics\_raw["resp\_key\_col"] = df\_metrics\_raw[

            ["offer\_package\_name", "offer\_package\_id"]

        ].apply(lambda x: x[0] + "##" + x[1], axis=1)

        # use pandas filter fn instead of lambda

        df\_metrics\_raw["scenarios"] = df\_metrics\_raw[

            ["resp\_key\_col", "scenario\_value", "lift"]

        ].apply(

            lambda x: {x[0]: {"scenario\_value": x[1], "lift": round(float(x[2]))}}, axis=1

        )

        return dict(

            df\_metrics\_raw[["kpi\_name", "scenarios"]]

            .groupby("kpi\_name")["scenarios"]

            .apply(list)

        )

(Pdb)

### up,down and where

where- prints the stack trace (the order of execution which lead to current line)

up- a line up (from the current line) the stack trace

down- a line down (from the current line) the stack trace

(Pdb) help up

u(p) [count]

        Move the current frame count (default one) levels up in the

        stack trace (to an older frame).

(Pdb) help down

d(own) [count]

        Move the current frame count (default one) levels down in the

        stack trace (to a newer frame).

(Pdb) help where

w(here)

        Print a stack trace, with the most recent frame at the bottom.

        An arrow indicates the "current frame", which determines the

        context of most commands.  'bt' is an alias for this command.

### Quitting or exiting

If you want to exit or abort the execution

(Pdb) help q

q(uit)

exit

        Quit from the debugger. The program being executed is aborted.

(Pdb) help exit

q(uit)

exit

        Quit from the debugger. The program being executed is aborted.

### step or s

if the current line in the debugger is a function,

ex: remainder = division(num1,num2)

if you want to go inside the function, then s will help you do that

### next or n

(Pdb) help n

n(ext)

        Continue execution until the next line in the current function

        is reached or it returns.

### return or r

(Pdb) help r

r(eturn)

Continue execution until the current function returns.

no matter where you are in the current function, if you want go to the end of the loop, you can just give type return or r

**this is especially helpful if you are inside a loop, pressing c** **or r** **will help you solve the problem**

## clear or cl

(Pdb) help clear

cl(ear) filename:lineno

cl(ear) [bpnumber [bpnumber...]]

With a space separated list of breakpoint numbers, clear

those breakpoints. Without argument, clear all breaks (but

first ask confirmation). With a filename:lineno argument,

clear all breaks at that line in that file.

Bpnumber means breakpoint number

### jump or j

(Pdb) help j

j(ump) lineno

Set the next line that will be executed. Only available in

the bottom-most frame. This lets you jump back and execute

code again, or jump forward to skip code that you don't want

to run.

It should be noted that not all jumps are allowed -- for

instance it is not possible to jump into the middle of a

for loop or out of a finally clause.

# miscellaneous:

* easy print formatting to slip variables in strings, this belo

a='this is a'

b='this is b'

print(f'{a},{b}')

* os.system will help you execute shell commands
  + os.system('date') # will give current date
  + os.system('notepad')#will open notepad
  + os.system(‘ls’)
* Disadvantage of python dynamic typing (dynamic typing -no need to specify type of variable while typing):

a,b=’hello’,7

>>> a+b

Traceback (most recent call last):

File “”””, line 1, in

a+b

TypeError: can only concatenate str (not “”int””) to str

In languages like C++ and Java, adding the two variables will result in a compilation error since one is a string and another is an integer.

In Python, however, this is syntactically correct. When we run this code though, it results in a TypeError.

* Encapsulation and abstraction help us by hiding details and protecting the consistency of our data, but we also need to pay attention to the interactions between our objects and functions. **When one function, module, or object uses another, we say that the one depends on the other**

- (underscore) will store the latest assignment or the latest or the latest result.

-- you can also use else block for try except block, for block, while block as well

-In Python, functions are first class objects that means that functions in Python can be used or passed as arguments.

--"in" and "not in" operators return true or false about existence of an element/value in an iterable

--dir(variable/method/library) will print all methods that can be applied with that variable

--isinstance('sai',str) will return true if first parameter datatype matches with second parameter

--def power(num):

return num\*\*2

lis=[0, 1, 2, 3, 4, 5, 6]

squared=list(filter(power,lis))#filter will remove the elements in the iterable and return elements which are viable with the logic,whereas map will return the value which the function will return for better understanding remove filter and place map.

--filter: applies a certain condition to items in the iterable and rules out elements which are not viable with the logic

even\_lst=list(filter(lambda x:(x%2==0),lst))#very usefull

--from functools import reduce

def sum(x,y):

return x+y

lis=[0, 1, 2, 3, 4, 5, 6]

squared=reduce(sum,lis)

print(squared) #o/p:21

reduce will return a single value so we will not use list as we use list(map()

--default argument in function assigns a value to a variable in that function, if any value is assigned in function call, then that value is used instead of default value

--map,filter or reduce will have 2 parameters parameter 1 is the function and parameter 2 is the iterable

--a file with name abc.py containing code is called a module and name of the module is abc,to import that module we simply write abc.py

--to copy the content of a file to string in one line use:

from pathlib import Path

file\_path=r"C:\Users\saivinil.pratap\Downloads\backup db poc\member db\member bkp.sql"

query = Path(file\_path).read\_text()

--to check the mime type of a file in python

import mimetypes

print(mimetypes.MimeTypes().guess\_type(<path of file>)[0])

-- there is no direct do while loop in python, but you can use the below structure:

while(True):

if(condition)#the condition for which you want to exit from the loop

break

--shelve in python:

import shelve#this works exactly like dictionary with a small difference as it can be stored and called from anywhere in the memory

shelfFile=shelve.open('mydata')#This creates mydata.bak mydata.dir mydata.dat files in the current folder(or current working directory) as the path passed is a relative one , if the path is absolute then the before mentioned 3 files will be created in the mentioned absolute paths

shelfFile['people i love']=['ANITHA','BADRI','LATHA','AKSHAY','NEERJA','MYSELF']#this is key value declaration

print(shelfFile['people i love'])#this will print all the values that are under the current key

shelfFile['people i hate']=['I HATE NO ONE']

print(shelfFile['people i hate'])

shelfFile['1']='1'

print(list(shelfFile.keys()))#will print all the list of keys that are under the current shelfFile object based on the path passed in sheshelve.open() method

print(list(shelfFile.values()))# this will return a list of elements which are declared on current shelfobject

shelfFile.close()

--

You can see which Python interpreter you're running by doing:

>>> import sys

>>> sys.executable

-to check the location of the package

import <package name>

<package name>.\_\_file\_\_

-to install package directly from jupyter

!pip install <packagename>

-to check version of python

from platform import python\_version

print(python\_version())

-environment error while installing packages can be removed if we try using admin mode

--decorators in python:

let us say we have some function in some class which is to be imported into the current class and you need to do changes to the mthod but not in the class from which it is imported but in the current class then we can use decorators

def div(a,b):

print(a/b)

# it will return the division of a/b but if you need to only get numbers > 1 we need to have the bigger number in the numerator so we can use decorators to alter the function as below,this is a very small introduction for decorators

def smart\_div(func):

def inner(a,b):

if a<b:

a,b=b,a

return func(a,b)

return inner

div=smart\_div(div)

div(2,4)

--variable arguments usage in python:

fav\_col=['pink','orange','green']

def fav\_col\_printer(\*col):

print('my fav colors are ' , ', '.join(\*col))

print('printing a list using \*',\*fav\_col)

fav\_col\_printer(fav\_col)

#here no matter how many colors you chose the code will run fine as it obeys variable argument usage.

--operator overloading:

#operator overloading through classes

class inti:

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

self.num=num

def \_\_add\_\_(self,other):

summ=self.num+other.num

print('the sum of 2 numbers is '+str(summ))

n1=inti(5)

n2=inti(6)

n1+n2

#here operator overloading is happening as \_\_add\_\_ is the default method of addition but we have over rided the method by calling it in the current class so if we add 2 objects it is calling the locally defined \_\_add\_\_ method this is called operator overloading in python in similar way we can override all the methods we obtain using dir(datatype) option

--method overriding

a=1

# print((a.\_\_str\_\_()))

print(a)

class inp:

def \_\_init\_\_(self,inti,inti2):

self.inti=inti

self.inti2=inti2

c=inp(1,2)

print(c)

class inti:

def \_\_init\_\_(self,inti,inti2):

self.inti=inti

self.inti2=inti2

def \_\_str\_\_(self):

# print(type(self.inti))

return '{} {}'.format(self.inti,self.inti2)

b=inti(1,2)

print(b)

print(b.\_\_str\_\_())

#when we try to print an object by default it will call(even if we call or not) the .\_\_str\_\_() and by default \_\_str\_\_() will return the address of the object, but in some exceptional cases if the object is storing an inbuilt datatype it will print the value the variable holds see the printing of variable a b and c in above example, this is an example of method overriding because the default \_\_str\_\_() method is getting overriden by the current declared method, THIS STRING METHOD SHOULD BE DEFINED IN SUCH A WAY THAT IT COVERS AND OUTPUTS ALL MEMBERS OF A CLASS, consider the below example.

-

class MyClass:

x = 0

y = ""

def \_\_init\_\_(self, anyNumber, anyString):

self.x = anyNumber

self.y = anyString

def \_\_str\_\_(self):

return "the Myclass contains {} as a number and {} as a string".format(self.x,self.y)

myObject = MyClass(12345, "Hello")

# even though the the below print statements are different they give the same output(if the str method is not overrided, if it is overrided repr method will give different results), because when an object is instantiated in the class, it will populate this str method involuntarily and if you print the object it will call str method involuntarily.

print(myObject.\_\_str\_\_())

print(myObject.\_\_repr\_\_())

print(myObject)

-but if the \_\_repr\_\_ method is overrided then the output for class.\_\_str\_\_(), and other print statements used in the below program return the same results

class MyClass:

x = 0

y = ""

def \_\_init\_\_(self, anyNumber, anyString):

self.x = anyNumber

self.y = anyString

def \_\_repr\_\_ (self):

return "the Myclass contains {} as a number and {} as a string".format(self.x,self.y)

myObject = MyClass(12345, "Hello")

#the below print statements will return the same values.

print(MyClass.\_\_str\_\_(MyClass(1,'23')))

print(myObject.\_\_str\_\_())

print(myObject)

print(str(myObject))

print(myObject.\_\_repr\_\_())

--method overloading: though we dont strictly have an method overloading function in python(as we cannot declare two methods with same name and one more other reason is python is dynamically typed language and you cannot have two methods with same name and same parameters and different datatypes as python determines the datatype dynamically and therefore no need of having two methods declared with same number of parameters and different data types) the closest we can come is with the example below or by using keyword arguments

ex:

from multipledispatch import dispatch

#passing one parameter

@dispatch(int,int)

def product(first,second):

result = first\*second

print(result);

#passing two parameters

@dispatch(int,int,int)

def product(first,second,third):

result = first \* second \* third

print(result);

#you can also pass data type of any value as per requirement

@dispatch(float,float,float)

def product(first,second,third):

result = first \* second \* third

print(result);

#calling product method with 2 arguments

product(2,3,2) #this will give output of 12

product(2.2,3.4,2.3) #

--stream methods return a value but the value which it stores will remain intact, for example if you call a method upper it will give a different result, but the value it stores stays the same

ex:

s='this is original string with small case letters'

print(s.upper(),s)#here s will not change

--print()#this will print a empty line

-- the difference between string and list is that

1:string is immutable whereas the list is not

ex:strp='hiiii',strp[3]='p' #will result in an error

--getting precision up to n decimal places

1:print('%.2f'%4.346)#output:4.35

2:print('{0:.8f}'.format(4))# will print 4.35

3:print(round(5,4)) # will print 5 ,

but

print(round(5.1654435,3))#will print upto 3 decimal places i.e, 5.165

?<any variable or method or function or package> will give its type,string,file location and docstring

-- if a python instruction evaluates to a single expression it is a expression or else it is a statement

-- interactive shell or (interpreter) evaluates one instruction at a time, file editor (such as notepad or any file generating program which can hold multiple instructions and save them as a python file with .py extension)

--

name=input()

if name:

print('name is entered')

else:

print('nothing is entered')

here if name is not entered it will go to else loop because empty string is treated as false by python

0,0.0,'' are treated as false by python

--None returns nothing(type it in interpreter and you can see by yourself)

a function without any return statement will return none

--main function in python will be something like this

if \_\_name\_\_== '\_\_main\_\_':

#enter code here

--

the below code will calculate ((input1^input2)+(input3^input4))

if \_\_name\_\_=='\_\_main\_\_':

print (pow(int(input()),int(input()))+pow(int(input()),int(input())))

--str\_list = ['A', 'C', 'F']

syntax for combinations: itertools.combinations(list\_of\_strings,count\_of\_strings)

list(itertools.combinations(str\_list,2))o/p:[('A', 'C'), ('A', 'F'), ('C', 'F')]

list(itertools.permutations(str\_list,2)o/p:[('A', 'C'), ('A', 'F'), ('C', 'A'), ('C', 'F'), ('F', 'A'), ('F', 'C')]#position starts with p and permutation starts with p and position also matters in permutations, that means ('A','C') is different from ('C','A')

--chr(unicode code point) will return symbol in string format of the character

print (chr(176))# and check output,other type convertrs are int,str,float,ord,hex,oct,list,tuple,dict,set,complex

--bin(number) will return binary from of number

ex:bin(7) will return '0b111'

--the first line of all python programs should be a shebang line.

begins with a #!

in windows it is:#! python3

in osx: #!usr/bin/env python3

in linux: #!usr/bin/python3

in windows python.exe will read the shebang line at the top of the python source code and will decide which version of python should run the script

--sys.argv will store the path of the python file(the file which is called for running) and all other command line arguments(parameters that are passed as inputs for program to execute) that are passed along with while running the file in list of strings format

--python supports batch file execution,a batch file will run lot of commands that contains in itself at once.(will have .bat extension)

--for \_ in range(5):

print('hi)

#will print hi for 5 times even if you did not use any iterable in for statement

--when you use an object as a default argument in function and if the function is called more than once and if you have some variables in def \_\_init\_\_(self): of the default object which is used as a parameter then they will not get initialized again and the old initialized/changed values (when it is called for the first time will be used again and this will lead to an error.

-- when input is passed to a program the value is treated as string initially, to cast it into the type we need programmers follow the conventional way mentioned below the datatype mentioned below is user specific:

input\_list=list(map(datatype,input().rstrip().split()))

my approach: input\_list=[int(i) for i in input().split()]

-- if you are operating on a variable which is declared in a for loop and incrementing or decrementing it in the same loop is a mistake

ex:if you want to print values form 1 to 10 then

for i in range(10):

i=8+i

print(i)#this will not give the same result

-- placing a r before a string will make the interpreter treat it as a raw one and when a string is raw it treats escape sequences as escape sequences and make its functionality work, if there is a single slash,it wont check what is its adjacent character and thereby avoids unexcpected escape seuence getting reflected

ex:

#here in the below example first is a raw string so a single slash is treated as single slash, but if the string is not a raw sting we have to mention double slash as an escape sequence for a slash is to mention it as a double slash.

print(r"C:\Users\saivinil.pratap\Desktop\PractitionerDataModel\_v0.xlsb.xlsm","C:\\Users\\saivinil.pratap\\Desktop\\PractitionerDataModel\_v0.xlsb.xlsm")

-- escape sequences help us to include characters in a string which are hard to type

-- when you try to divide a number by zero, it raises an error (ZeroDivisonError) and there are many such errors and an error will cause the program to crash and we have to avoid the crash because it will prevent the program from running,to prevent errors we can use exception statements,in layman terms python doesn't know how to carry on with improper data and it also does not want to run program with improper data

ex:

num=input()

try:

if int(num)>=0:

print('dividing 20 with input number gives',20/int(num))

else:

print('please enter positive number')

except ZeroDivisionError:

print('you cannot divide a number by zero')

except:

print('please enter input in number format')

-int('05') will print 5

-x, y, z, n = (int(input()) for \_ in range(4)) #crisp code to accept for 4 consecutive inputs

-print('Hello',first,last+'!','You just delved into python.')#here , will replaced by an empty space ' ', whereas + will make the concatenation next to it without any space in between, as the name represents in english when you have comma between the wods you have to take a pause and then continue reading,the same way comma will use a space similar to pause in english language, and + in math will add 2 values,in the same way python uses + between strings to concatenate them immediately

-batch files achieves the automation, you can include a syntax

syntax:@py <path of the script to be executed> %\*

the star at the end takes the command line arguments as input i.e, sys.argv[0] will be the path of the script and the other inputs after that seperated by space are sent as commandline arguments

after you save the file as a batch file (with .bat extension) , then declaring that file containing folder in the path variable of system variable. so that on pressing windows+r we can directly type the name of the bat file which in turn executes the python script inside it.this saves a lot of time when we want to run scripts periodically

* To find the location of installed package, use interpreter, then import <package>, then <package>.\_\_file\_\_
* When you give export FLASK\_APP=<script\_name.py> the script name is treated as package, when you try to give **flask run** to run the app, \_\_name\_\_ will store the name of the script , because the script is being treated as package, if you give python\_script.py **here, the script is being run as the main program** to run the app, then \_\_name\_\_ will store \_\_main\_\_ and then the code under \_\_name\_\_==’\_\_main\_\_’ will get executed
* when you are facing package not found error even after installing them, then the interpreter is not able to find it, then you can check the python version that is set, and then install package in that version
  + syntax: <python version> resolve pip install <package name>
* setting python 3.5 as default interpreter on centos:
  + sudo ln -fs /usr/bin/python3.5 /usr/bin/python
* when you activate both pip and conda virtual environment, afaik the environment which is activated later will be active until it is deactivated. You can check this by checking which python environment is active by giving “which python” command
* --Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped.
* --str="Text may contain stop words like ‘the’, ‘is’, ‘are’. Stop words can be filtered from the text to be processed. There is no universal list of stop words in nlp research, however the nltk module contains a list of stop words.

You can add your own Stop word. Go to your NLTK download directory path -> corpora -> stopwords -> update the stop word file depends on your language which one you are using. Here we are using english (stopwords.words(‘english’))."

str.count("the")

## types of casings

snake casing: each space is replaced by underscore

ex: **foo bar** is converted as **foo\_bar**

camel casing: each words starting letter is capitalized and spaces are removed, except the first letter

ex: **foo bar** is converted as or  **foo\_Bar**

pascal casing: each words starting letter is capitalized and spaces are removed, including the first letter

ex: **foo bar** is converted as **Foo\_Bar**

## creating directory in python using pathlib

import pathlib

output\_dir = "/opt/ml/processing/evaluation"

pathlib.Path(output\_dir).mkdir(parents=True, exist\_ok=True)

## StringIO in python

This method reads a string and converts all expected types correctly and converts them in to a dataframe #have to play with it and check

import pandas as pd

from io import StringIO

input\_data = "1,2,3,4,5,6.0,a"

df = pd.read\_csv(StringIO(input\_data), header=None)

print(type(list(df[6])[0]))

## Ducktyping

if an object walks like a duck,quacks like a duck,swims like a duck then the "object" is a duck, this ducktyping happens in python, i.e, if a number is assigned to a variable then the variable can perform all the tasks an integer can perform similarly if a string is assigned to the same variable then the variable can perform methods related to string object such as 'isdecimal','isdigit','isidentifier'. PYTHON OBEYS DUCK TYPING.

## logging in python

-- logging data in program to a text file while running

'''logging is divided in to 5 categories and it has some priority

below are the 5 categories:

1:logging.DEBUG

2:logging.INFO

3:logging.WARNING

4:logging.ERROR

5:logging.CRITICAL

HERE THE ORDER OF PRIORITY IS DEBUG being the least and CRITICAL being the highest, this is useful when we disable it when we disable a log of particular order,then the priority logging messages will be disabled provided that they have less priority than the current priority logging category

syntax:logging.disable(category)

'''

# logging.disable(logging.ERROR) will ignore ERROR,WARNING,INFO,DEBUG categories

import logging

# logging.basicConfig(filename=r"C:\Users\saivinil.pratap\Desktop\log.txt",level=logging.DEBUG,format='%(asctime)s - %(levelname)s - %(message)s') #this syntax will save the log message in the specified destination as text file

logging.basicConfig(level=logging.DEBUG,format='%(asctime)s - %(levelname)s - %(message)s') #this will just print log messages on the console

logging.debug('start of the program')

def factorial(n):

logging.debug('start of factorial (%s)' % (n))

total=1

if n < 0:

logging.critical('entered value is %s we cant calculate value of a negative number',(n))

return None

for i in range(1,n+1):

total \*=i

logging.info('i is %s,total is %s' % (i, total))#notice there are no commas and parameters are in ()

logging.info('return value is %s'%(str(total)))

return total

print(factorial(-1))

logging.debug('end of program')

# pending formattings

arrange [import heading](#_import:) properly in to sub headings

# topics to learn:

from collections import Counter, OrderedDict

str.maketrans

--check os and sys functionalities completely in python(incomplete)

--recursion problems

--WHY IS MAIN SURROUNDED BY \_\_

--https://www.geeksforgeeks.org/decorators-in-python/

# Hashtags used

* #need clarity : to dig deeper and to understand
* #last stopped point : reading last stopped at this point
* #have to play with it and check : when you know some piece of code and not sure of how it works and you want to spend some time on it later on.
* #incomplete: need to fill pending information

20/09/22