NUMPY

NumPy is a Python library used for working with arrays, linear algebra, fourier transform, and matrices.
A numpy array is similar to a list
NumPy stands for Numerical Python and it is an open source project.
The array object in NumPy is called **ndarray**

Linspace

A useful function for plotting mathematical functions is linspace. Linspace returns evenly spaced numbers over a specified interval.

numpy.linspace(start, stop, num = int value)

start : start of interval range
stop : end of interval range
num : Number of samples to generate.

Syntax	np.array([[1,2,3,4] , [2,3,4,5]])	l = [[1,2,3,4] , [2,3,4,5]]	Indexing of list elements is similar to indexing of array elements. Array involves dimensions which makes it different from list
Output	array([[1, 2, 3, 4], [2, 3, 4, 5]])	[[1,2,3,4] , [2,3,4,5]]	

Array is also called nested sequence.

NUMPY ARRAY			
Sr.No.	**Syntax/command**	**Output**	**Remarks**
1	import numpy as np		To activate numpy from library
2	np.array(l)	array([1, 2, 3, 4, 5, 6])	To convert list l to array
3	l1 = [4,5,6,7,"sudh', 45.56",True] np.array(l1)	array(['4', '5', '6', '7', "sudh', 45.56", 'True'], dtype='<U12')	While converting list with all different kinds of values, array will change all values to string format. U12 in internal representation for numpy array.
4	np.array([[1,2,3], [4,5,6]])	array([[1, 2, 3], [4, 5, 6]])	List inside a list will give 2 dimensional array
5	np.array([[[1,2,3], [4,5,6], [7,8,9]]])	array([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])	List inside a list will give 3 dimensional array
6	a3.ndim	3'	Will indicate dimension of the array.
7	a3.size	9	Will give the number of elements in an array. A 3x3 array will yield value 9.
8	a1.shape	(2, 3)	Will indicate matrix size of the array. (2, 3) means 2x3 matrix.
9	a3.shape	(1, 3, 3)	Means we have 1 nos 3x3 matrix.
10	np.random.randint(2,50)	11	Will indicate a random integer from 2-50
11	np.random.randint(2,50,(3,4))	array([[49, 37, 42, 9], [24, 38, 5, 21], [9, 8, 25, 9]])	Will generate a 3x4 matrix with random values ranging from 2-50
12	np.random.rand(2,3)		Generates data with random mean and random standard deviation.
13	a4 = np.random.randn(3,4)		Generates random data of normal distribution with mean = 0 and standard deviation = 1. Avg of each of the column in the matrix will be zero
14	a4.reshape(2,6)		It will crop or reshape the 3x4 matrix to 2x6 matrix and to execute the command total no of elements should be same.
15	a4.reshape(12,-1)		Here -1 will auto calcutate value in its place based on matrix size and total elements. Any negative value can be used.

16	a4.reshape(2,3,2)		Will convert 2 dim array to 3 dim provided total no of elemnts are same in both cases i.e 2 nos of 3x2 matrix. (2*3*2=12)
17	a4.reshape(1,1,1,1,1,1,2,3,2)		The no of 1's within the bracket will increase the dimensions of the arrary i.e it will add the same no of square brackets as the no of 1's. Rest is 2*3x2 matrix
18	a1[2:6]		**Slicing operation** : It will extract data between index 2 to 6 (excluding upper limit) from the array.
19	a1[::-1]		It will reveerse the entire array.
20	a2[:,1:] or a2[:,[1,2]] or a2[[0,1] , 1:]		To extract rows and columns in a matrix array. The first part indicates rows and then the columns.
21	a5[a5>40]		Will give array data of all elements greater than 40
22	a5[0,1] = 99		Will change array element at 0,1 to 99
23	a6 * a7		Will execute multiplication element wise based on the coordinates.
24	a6 @ a7		**Matrix multiplication**
25	a6 + 100		It will add 100 to each and every element of a6
26	a6 * 2		It will multiply 2 to each and every element of a6
27	a6/0		It will divide 0 to each and every element of a7. It will give warning but still give the inf as ans.
28	a6**3		It will give power of 3 to each and every element of a6
29	np.zeros((4,4))		It will give array with all elements 0
30	np.ones((4,4))		It will give array with all elements 1
31	a9 + np.array([1,2,3,4])		**Broadcasting Operation**: Will add the array row wise to the rows of a9 array

32	np.array([[1,2,3,4]]).T + a9		T - transpose will switch the array vertically or horizontally provided you give it two dimension i.e additional brackets. And will carry out addition operation column wise now. **(Broadcasting operation)**
33	a6.T		Will interchange rows to columns and columns to rows.
34	np.sqrt(a5)		Will give square root of each and every element of a5
35	np.exp(a5)		Will give exponential of each and every element of a5
36	np.log10(a5)		Will give log value of each and every element of a5
37	list (range(0,10 ,2))	[0, 2, 4, 6, 8]	Perform list operation indicating values between 0 to 10 with a jump of 2. **Decimal values cannot be used for range function**
38	np.arange(10)	array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])	Similar to range function except that arange returns an array of range values
39	np.arange(0,10,2.5)	array([0. , 2.5, 5. , 7.5])	Used when jump size or values are in decimals.
40	np.linspace(2,3,num=50)		Will give array of 50 elements with values between range 2-3
41	np.linspace(2,3,num=50,retstep=True)	array([sample], step)	**Retstep command:** If True, return (`samples`, `step`), where `step` is the spacing between samples.
42	np.logspace(2,3,num=4,base=10)	array([100. , 215.443469 , 464.15888336, 1000.])	Will give log values to the base 10 between numbers 2 -3
43	np.eye(5)	array([[1., 0., 0., 0., 0.], [0., 1., 0., 0., 0.], [0., 0., 1., 0., 0.], [0., 0., 0., 1., 0.], [0., 0., 0., 0., 1.]])	Will give identity matrix of size 5x5 with diagonal elements as one and all other elements zero
43	a = np.array([3,4,5,6] , ndmin= 3)	array([[[3, 4, 5, 6]]])	It will return 3 dimensional array
43	arr.ndim	2	It will return dimension of the array
43	np.random.randint(1,10,(4,4,2))		It will generate 4 nos 4*2 array with random values between 1-10
43	print(np.__version__)		Checking NumPy Version

43	c = np.array([34,44,5,23,11,89,9]) select = [0,1,2,3] d = c[select] d	array([34, 44, 5, 23])	We can use the list as an argument in the brackets. The output is the elements corresponding to the particular indexes:
43	c = np.array([34,44,5,23,11,89,9]) select = range(4) c[select] = 1000 c		It will take select list as argument and replace those indexes with number 1000.
43	u = np.array([1, 0]) v = np.array([0, 1]) z = np.add(u, v) z	array([1, 1])	**Array addition**
43	a = np.array([10, 20, 30]) b = np.array([5, 10, 15]) c = np.subtract(a, b) c	array([5, 10, 15])	**Array subtraction**
43	x = np.array([1, 2]) y = np.array([2, 1]) z = np.multiply(x, y) z	array([2, 2])	**Array multiplication**
43	a = np.array([10, 20, 30]) b = np.array([2, 10, 5]) c = np.divide(a, b) c	array([5., 2., 6.])	**Array division**
43	X = np.array([1, 2]) Y = np.array([3, 2]) np.dot(X, Y)	7	**Dot product = 1*3+2*2=7**
43	u = np.array([1, 2, 3, -1]) u + 1	array([2, 3, 4, 0])	Adding Constant to a Numpy Array
43	np.pi x = np.array([0, np.pi/2 , np.pi]) y = np.sin(x) y	array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])	Using pie values in numpy
43	np.linspace(-2, 2, num=5)	array([-2., -1., 0., 1., 2.])	A numpy array within [-2, 2] and 5 elements
43	arr1 = np.array([1, 2, 3]) print(arr1) for x in arr1: print(x)	[1 2 3] 1 2 3	**Iterating 1-D Arrays**
43	A[1, 2]	Same ans	Extracting values from the array
43	A[1][2]		
43	X = np.array([[1, 0], [0, 1]]) Y = np.array([[2, 1], [1, 2]]) Z = X + Y Z	array([[3, 1], [1, 3]])	**Matrix addition**

43	Y = np.array([[2, 1], [1, 2]]) Z = 2 * Y Z	array([[4, 2], [2, 4]])	**Multiplying a matrix by a scaler**
43	Y = np.array([[2, 1], [1, 2]]) X = np.array([[1, 0], [0, 1]]) Z = X * Y Z	array([[2, 0], [0, 2]])	Element-wise product of the array X and Y
43	C.T		Transposed of C
43	A = np.array([[0, 1, 1], [1, 0, 1]]) B = np.array([[1, 1], [1, 1], [-1, 1]]) Z = np.dot(A,B) Z	array([[0, 2], [0, 2]])	**Matrix multiplication** with the numpy arrays A and B

Pandas: DataFrame and Series

Pandas is a popular library for data analysis built on top of the Python programming language.
Pandas generally provide two data structures for manipulating data, They are:

DataFrame
Series

A **DataFrame** is a **two-dimensional data structure**, i.e., data is aligned in a tabular fashion in rows and columns.

A Pandas DataFrame will be created by loading the datasets from existing storage.
Storage can be SQL Database, CSV file, an Excel file, etc.
It can also be created from the lists, dictionary, and from a list of dictionaries.

A **Series** represents a **one-dimensional array** of indexed data. It has two main components :

1. An array of actual data.
2. An associated array of indexes or data labels.

The index is used to access individual data values. You can also get a column of a dataframe as a Series.
You can think of a Pandas series as a 1-D dataframe.

Concatination operation of two datasets:

1. Vetically
2. Horizontally

Merge	Based on columns/features	Default = 'inner'
Join	Based on indexes	Default = 'left'

Series	One dimensional array
Dataframe	Two dimensional array

For selecting row labels	Axis = 0
For selecting column names	Axis = 1

* 0 or 'index': apply function to each column.
* 1 or 'columns': apply function to each row.

Pandas and Numpy are used for data manipulation

PANDAS PROFILLING			
Sr.No.	**Syntax**	**Output**	**Remarks**
1	Import pandas as pd		
	df = pd.read_excel('Attribute DataSet.xlsx')		
	type(df)	pandas.core.frame.DataFrame	
2	df = pd.read_excel(r'D:\Data Science\Ineuron\Main course\Python\Pandas\data fsds - 20221122T052032Z-001\data fsds\Attribute DataSet.xlsx')		Document in the form of CSV, excel, JSON format can be loaded by either uploading in directory or by using location link from system.
3	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1")		You can specify sheet name or index to open particular sheet in the excel document.
	df1 = pd.read_excel(r'Attribute DataSet.xlsx', 0)		
4	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1", header=1)		To choose a different header row for pandas profiling. Pandas uses first row as deault header.
5	df1 = pd.read_excel(r'Attribute DataSet.xlsx', sheet_name="Sheet1", header=None , names=['A','B','C','D','E','F','G','H','I','J','K','L','M','N'])		Use custom column names if it is not present in the dataset.
6	df.head(2)		It will show top rows as per arg parsed.
7	df.tail()		It will show bottom most rows as per arg parsed.
8	df3 = pd.read_csv(r'haberman test.txt',sep ='@')		If Delimiter other than comma in csv, the same shall be specified to get individual columns.
9	pd.read_csv('https://raw.githubusercontent.com/Opensourcefordatascience/Data-sets/master/blood_pressure.csv')		To load data from github, use its raw format from github repository to load dataset.
10	a = pd.read_html('https://www.basketball-reference.com/leagues/NBA_2015_totals.html')		To extract table from the website. It will return a list in case of multiple tables. Using index of the particular table will return a dataframe.
	a[0]		
11	js=pd.read_json('https://api.github.com/repos/pandas-dev/pandas/issues')		Read json file in pandas using json viewer to get idea about the structure
	js.columns		
	js['user']		
12	df.to_csv('D:\Data Science\Ineuron\Main course\Python\Pandas\Pandas_practice.csv', sep='#')		To save file at required location and using # instead of comma as delimiter
13	df.to_csv('D:\Data Science\Ineuron\Main course\Python\Pandas\Pandas_practice1.csv', sep='#',index=False)		Index will be removed before saving the file
	pwd		Return the current working directory path.
	Analysising the data		
14	df.head()		It will show top 5 rows of dataset
15	df.tail()		It will show bottom 5 rows of dataset
16	df.columns		It will return list of column names
17	df.dtypes		It will give data type of each of the columns/feature in the dataset
18	df['profit']	Same output	It will return data under profit in series format
	df.profit		
19	df['profit','year']	Error	You cannot pass 2 argument without additional brackets

20	df[['profit','year']]		It will retun dataframe of 2 columns. You have to pass the columns in list parameter.
21	df[['profit']]		It will retun dataframe of single column
22	df[['order_id','product_id','quantity']]		It will showcase all these columns
23	df.describe()		It will give statistical data of only numerical columns in the dataset
24	df.dtypes == 'object'		It will return boolean True for all object columns and false for numerical columns.
25	df.dtypes[df.dtypes == 'object']		It will return series format of the object columns
26	df.dtypes[df.dtypes == 'object'].index		It will give list of all the object columns in the dataset.
27	df[df.dtypes[df.dtypes == 'object'].index]		It will return dataframe of object columns
28	df[df.dtypes[df.dtypes == 'object'].index].describe()		It will give statistical data of only object columns in the dataset
29	df[df.dtypes[df.dtypes == 'float64'].index]		It will filter out only float type columns in the dataset with dataframe
30	df['order_id'][1:40:2]		It will return series data from index 1 to 40 with a jump of 2 **(slicing operation)**
31	df['category1'] = "sudh"		It will add a new column "catagory1" and assign the same value to all rows in the column
32	df['order_id'].isnull()		It will return series of boolean true and false values over each value of the column.
33	df[df['profit'] == max(df['profit'])]		To filter out row who has got max profit
34	df[df['profit'] == max(df['profit'])]['customer_name']		Name of customer who has got max profit
35	df[df['country'] == 'Sweden']		It will give dataframe of all sweden countries
36	len(df[df['country'] == 'Sweden'])		It will give count of number of rows in the new dataset.
37	df[df['shipping_cost'] > 80]['country']		Series format of all countries where shipping cost is > 80
38	df[(df['shipping_cost'] > 100)& (df['profit']< 10)]		Dataframe with rows satisfying both the conditions
39	df[['profit','customer_name']].max()	profit 8399.976 customer_name Zuschuss Donatelli dtype: object	Profit value along with name of customer with max profit. The sequence dosent matter.
40	df['converted_order_date'] = pd.to_datetime(df['order_date'])		Convert date from string format to datetime format and create a separate column for it
41	df['order_date_year'] = df['converted_order_date'].dt.year		It will extract the year from date and create a separate column for year
42	df['order_date_month'] = df['converted_order_date'].dt.month		It will extract the month from date and create a separate column for month
43	df['order_date_month'].value_counts()		It will perform groupby count operation
44	df['cost_to_comapny'] = df['discount'] + df['shipping_cost']		It will create new column by adding values from 2 other columns
45	df[df['cost_to_comapny'] == max(df['cost_to_comapny'])]['product_name']		Find the product where cost to company is max

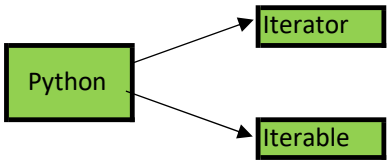
46	df.drop('cost_to_comapny',axis=1,inplace=True)		To drop a **column** from dataset. In order to retain changes use inplace=True. Or you can reassign the changes to retain the same.
47	df.drop(1, inplace=True)		It will drop the **1 index row** and due to inplace=True it will retain the changes
48	df.loc[[2,3]]		It will filter out record from 2 and 3 index rows or labels
49	df.loc[0:4,['order_id','order_date','ship_date']]	Same output	It will filter data with reference to rows and columns
50	df[["Dress_ID", "Style", "Price"]].loc[0:4]		
51	df.iloc[0:4 , 0:3]		It will filter data with reference to default index of rows and columns
52	df.dtypes[(df.dtypes == 'float64') |(df.dtypes == 'int64')]		It will show all columns which are either float type or int type
53	df2 = df[df.dtypes[(df.dtypes == 'float64') |(df.dtypes == 'int64')].index]		Dataframe of only numerical columns
54	df3.dropna()		It will drop rows even if it has a single NaN value. Default axis = 0. axis = 0, or 'index' : Drop rows which contain missing values.
55	df3.dropna(axis=1)		It will drop columns even if it has a single NaN value. Axis = 1, or 'columns' : Drop columns which contain missing value.
56	df.dropna(how='all')		Drop the rows where all elements are missing.
57	df.dropna(thresh=2)		Providing condition for NaN values to drop the column
58	df.dropna(subset=['name', 'toy'])		Define columns to look for missing values. If you are dropping rows these would be a list of columns to include.
59	df3.fillna(value=4)		It will fill all NaN values with 4
60	df3.fillna(value=df3['profit'].mean())		It will fill all NaN values with mean of profit
61	df.groupby('order_date_year')['profit'].mean()		This will give avg of profit each year
62	df.groupby('order_date_year')['sales'].mean()		Avg sales
63	df.groupby('order_date_year')['shipping_cost'].mean()		Avg shipping cost
64	df.groupby('order_date_year')['discount'].mean()		Avg discount
	df['sales'] = df.sales.str.replace(',' , "").astype(int)		Convert str to int
	dfs3['Pin Code'] = dfs3['Pin Code'].astype(int)		
65	df.groupby('order_date_year')['sales'].sum()		Sum of sales each year
	ADVANCE		
18	df = pd.DataFrame(data,index=(4,5,6,7))		It will change the indexing of rows in the dataset
19	df.loc[5:6]	Both will show same result	It will extract data between the two row indexes that are labeled.
20	df.iloc[1:3]		It will extract data between the two row indexes that are default indexes
	df.iloc[0:5,2:7]		Data between (rows,columns) indexes.
21	pd.concat([df,df1])		Performing concatination operation **horizontally** (where axis=1)
	pd.concat([df,df1] , axis=1)		
23	pd.concat([df2,df3], axis=0)		Performing concatination operation **vertically** of datasets with same column names (where axis=0)

24	pd.merge(df4,df5)		It will filter out only that data which is common
25	pd.merge(df4,df5 , how ='left')		Similar to MySQL where we perform left join
26	pd.merge(df4, df5 , how = 'right' , on = 'emp_id')		Perform right joining operation based on certain column labels which is common between the 2 dataset.
27	pd.merge(df6,df7 , left_on='emp_id1' , right_on='emp_id2',how = 'inner')		Similar to MySQL where we perform inner join
28	pd.merge(df8,df9 , on = ['emp_id' , 'salary'])		it will merge the two dataset based on the two columns and only data which is common in both columns will be shown.
29	df10.join(df11)		Perform joining operation based on indexes

30	def profit_flag(a) :
	  if a >= 0 :
	  return 'positive'
	  else :
	  return 'negative'
			defiing functions and using it to create a separate column

31	df_sales['flag_profit'] = df_sales['profit'].apply(profit_flag)		
32	df_sales['len_cust_name'] = df_sales['customer_name'].apply(len)		Apply shall take argument in function
33	df_sales['square_quantity'] = df_sales['quantity'].apply(lambda a : a**2)		Applying lambda function

Some theory on python



In type of data i.e list,tuple,str,int for loop first tries to convert an iterable into an iterator. Data will be first available in the form of a iterable.
Only if data is iterable it can be converted to iterator.

Eg. Consider a list = [1,2,3,4]
then the list object is iterable but not in iterator form

How for loop internally works ?
In case of for loop:

If the item is iterable, the function first tries to convert iterable to iterator using **iter()** command.
It then uses **next()** command to extract each and every data.

Not all kind of data is iterable. Therefore if the object is not iterable you will not be able to convert those data to iterator.

list, tuple, Set, dict, Str objects by default are iterable so you can convert it to iterator using iter() command.

For loop uses **len()** command to stop the function at the end of object.

How does the range function work?
In case of range function:

it tries to generate a data set for us but the output is not directly displayed.
Yield command in place of print will store data in the form of generator object and will not give the output directly. same as in case of range function.
Therefore to convert any kind of function to a generator function use yield command.

Why is yield command used ?
When we define a function with return command and input very big value, the CPU in your system will take quite a long time processing to create those dataset and give the output and the laptop may even get hanged.

Whereas using yield command will give the output of each loop from the time the code runs and works in background to calculate and compute all the balance data without much affecting the processing power of CPU .

File operation

ls	command will open the default directory in drive C.
f = open("aditya1.txt")	will give an error if there is no file with that name in the directory
f = open("aditya1.txt",'w')	will create a new file in the directory with that name where 'w' means write
	r' means read, 'w+' means read and write, 'r+' means read and write.
f.write("this is my first file operation")	This will open the file and enter the data into the it to save the same.
f.close()	This command will close the file. With ls command if you open the
	file the input data size will reflect automatically

This operation is useful to save the data which you are temporirily holding in your juputer notebook

l = [1,2,3,4,5]	To enter list objects into a text file via python
f = open("text1.txt",'w')	
f.write(str(l))	In order to write list object you need to first convert it to a string variable
f.close()	You need to close the file for the data to reflect in the wordpad

f2 = open("text.adit",'w')	No matter in what format you save the file, it will always save in string format and will open in wordpad.

%%writefile text1.txt **this is my first python program to write into a file in a different way**	This will ***overwrite*** the text in the already existing file

File read operation	
f = open("text1.txt",'r')	It will display what ever data that is there in the file.
f.read()	Performing read operation the second time will not yield any dataas the first read operation has exhausted all the data and the pointer is at the end of very last object and there is no data after the last object
f.read()	
f.seek(0)	Will mover the pointer at the very same index in a string
f.read()	Only then read operation will yield data from start of the string. You can also provide index as per requirement to read the data.
f.tell(0)	It will show the pointer location in file.
f.read(15)	It will show the data upto the 15th index
f.readline()	will show one single line from the pointer
f.readlines()	will show all the lines present in the file.

Appending data in file operation	
f2 = open("text1.txt",'a')	a' stand for appending the data or else everytime
f2.write("ada dsaad")	it will overwrite the data.
f.name	will show the name of the file.
f.closed	to check whether the file is closed or not.
f.mode	will show a/r/r+/w/w+ mode

Mapping	**=map(function,iterable)**	It will implement the function operation to each and every element of the iterable.

Additional modes

It's fairly ineffecient to open the file in a or w and then reopening it in r to read any lines. Luckily we can access the file in the following modes:

r+ : Reading and writing. Cannot truncate the file.
w+ : Writing and reading. Truncates the file.
a+ : Appending and Reading. Creates a new file, if none exists.
tell() - returns the current position in bytes
seek(offset,from) - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2

corresponding to beginning, relative to current position and end

File operation			
Sr.No.	**Syntax/command**	**Output**	**Remarks**
1	l = [1,2,3,4,5] b = iter(l) next(b)	1	If you pass next() command directly on l it will give error saying list object is not a iterator. Using iter() command you can convert the list and perform next() command.
	next(b)	2	
	next(b)	3	
	next(b)	4	
	next(b)	5	
	next(b)	Error	It will give an error saying list object is not iterable.
2	def square_func(i): x = [j*j for j in range(i)] print(x)		when you enter commands within square brackets to directly convert data to a list - then it is called **"List comprehension"**
3	def square_func1(i): x = [j*j for j in range(i)] return x	[0, 1, 4, 9]	Return command will give the output of x directly.
	square_func1(4)		
4	def square_func2(i):		Whereas if you replace it with yield inplace of return it will not give the output directly.
	x = [j*j for j in range(i)]		
	square_func2(4)	Error	
5	next(square_func2(4))	[0, 1, 4, 9]	**Yield command** will make the entire data available as a iterator.

6	def fib_series1(n):
	a = 1
	b = 1
	l = []
	for i in range(n):
	l.append(a)
	a,b = b,a+b
	return l
	fib_series1(7)
		[1, 1, 2, 3, 5, 8, 13]	Function to get fibonacci numbers

7	**from functools import reduce**	36	Reduce function uses successive a,b values to calculate the sum() of the iterable. Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5)
	reduce(lambda a , b : a+b , l)		
8	list(filter(lambda x : x%2 ==0, l))	[2, 4, 6, 8]	Filter function will extract the data from iterable w.r.t function mentioned.

9	def even_func(n):
	if n%2 == 0:
	return True
		[2, 4, 6, 8]	Alternate to above code. Filter function works on the condition of True/False

	list(filter(even_func,l))		

10	import urllib.request # url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/data/example1.txt' # filename = 'Example1.txt' # urllib.request.urlretrieve(url, filename)		Download data

	from pyodide.http import pyfetch
	import pandas as pd
	
	filename = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%204/data/example1.txt"
	
	async def download(url, filename):
	  response = await pyfetch(url)
	  if response.status == 200:
	  with open(filename, "wb") as f:
	  f.write(await response.bytes())
	
	await download(filename, "Example1.txt")		

11	with open(example1, "r") as file1:
	  FileContent = file1.read()
	  print(FileContent)
			A Better Way to Open a File is using the with statement. The code will run everything in the indent block then close the file object.

12	with open(example1, "r") as file1:
	  print(file1.read(4))
	  print(file1.read(4))
	  print(file1.read(7))
	  print(file1.read(15))
			Once the method .read(4) is called the first 4 characters are printed. If we call the method again, the next 4 characters are printed. And so on.

13	with open(example1, "r") as file1:
	  print(file1.readline(20)) # does not read past the end of line
	  print(file1.readline(20)) # Returns the 20 chars of next line
			We can also pass an argument to readline() to specify the number of charecters we want to read in each line

14	with open(example1,"r") as file1:
	  i = 0;
	  for line in file1:
	  print("Iteration", str(i), ": ", line)
	  i = i + 1
			Iterate through the lines

	Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]
	with open('Example2.txt', 'w') as writefile:
	  for line in Lines:
	  print(line)
	  writefile.write(line)
			With \n it will write each sent on a new line

	with open('Example2.txt','r') as readfile:
	  with open('Example3.txt','w') as writefile:
	  for line in readfile:
	  writefile.write(line)
			It is used to copy data from readfile to write file

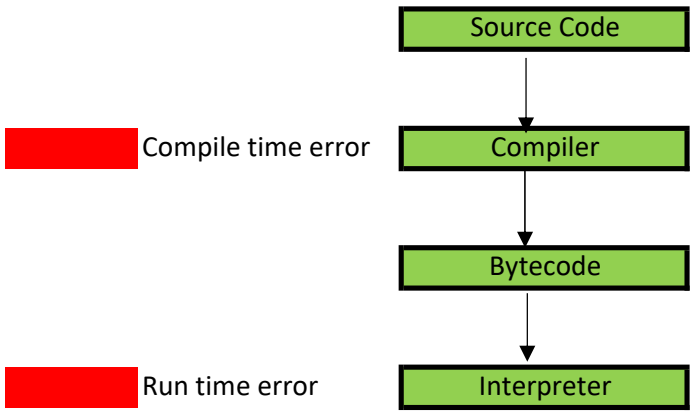
EXCEPTION HANDLING

An **exception** is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

A **try except** will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception

There are 2 types of error that a code or a program faces

1. Compile time error — Basically syntax error
2. Run time error — Error at interpreter level.



Eg. Code to a two variables

```
a = 10
b = "s"
c = a + b
print(c')
```

Type error : Cannot add int and str type
This is run time error

Once run the code will give exception saying cant add int and str
Finding the isuue is called exception handling.

Compiler will give error in case of missing words/characters eg print(c
or case

Exception ==== Run time error

Handling the error

Exception Handling

Why do we do exception handlng ?

Because of two reasons

1. When there is issue in one part of code, it will to help run code excluding that part of code.

```
Eg a = int(input("enter first number"))
   b = int(input("enter second number"))
   c = a/b
   return c
   d = a*b
   print(d)
   enter first number10          It is run time error
   enter second number0          It will give error for division but it does not give
                                 output for multiplication even when it is correct.
```

2. It error which pops out, its interface is not user friendly.

Catching an Exception

Python tries to execute the code in the try block. In this case if there is any exception raised by the code in the try block, it will be caught and the code block in the except block will be executed. After that, the code that comes after the try except will be executed.

Try Except Specific

Try Except Else and Finally

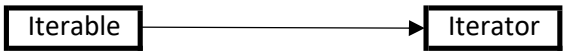
Iterators and Generators

Iterators:

• An **iterator** is an object that manages an iteration through a series of values. If variable, i, identifies an iterator object, then each call to the built-in function, next(i), produces a subsequent element from the underlying series, with a StopIteration exception raised to indicate that there are no further elements.

• An **iterable** is an object, obj, that produces an iterator via the syntax iter(obj).

list, tuple, and set, qualify as iterable types



data = [1, 2, 4, 8]

iter(data)	Will convert iterable to iterator
next(data)	extract subsequent data
StopIteration exception	When iterations are complete

Generators

A **generator** is implemented with a syntax that is very similar to a function, but instead of returning values, a yield statement is executed to indicate each element of the series

Keywords in exception handling
While true
try
except

else
raise
finally

Comprehension Syntax

Comprehension syntax is to produce one series of values based upon the processing of another series.

[k k for k in range(1, n+1)] list comprehension
{ k k for k in range(1, n+1) } set comprehension
(k k for k in range(1, n+1)) generator comprehension
{ k : k k for k in range(1, n+1) } dictionary comprehension

Packing and Unpacking of Sequences

data = 2, 4, 6, 8

results in identifier, data, being assigned to the tuple (2, 4, 6, 8). This behavior is called **automatic packing** of a tuple.

return x, y

it will be formally returning a single object that is the tuple (x, y).
As a dual to the packing behavior, Python can automatically **unpack a sequence,**

Simultaneous Assignments

The combination of automatic packing and unpacking forms a technique known as **simultaneous assignment**, whereby we explicitly assign a series of values to a series of identifiers,

x, y, z = 6, 2, 5

In effect, the right-hand side of this assignment is automatically packed into a tuple, and then automatically unpacked with its elements assigned to the three identifiers on the left-hand side.

all of the expressions are evaluated on the right-hand side before any of the assignments are made to the left-hand variables. This is significant, as it provides a convenient means for swapping the values associated with two variables:

j, k = k, j

j will be assigned to the old value of k, and k will be assigned to the old value of j.

A swap typically requires more delicate use of a temporary variable, such as

temp = j
j = k
k = temp

Scopes and Namespaces

Top-level assignments are typically made in what is known as **global scope**.
Assignments made within the body of a function typically have scope that is **local**

Each distinct scope in Python is represented using an abstraction known as a **namespace**.
A namespace manages all identifiers that are currently defined in a given scope.

dir( ) reports the names of the identifiers in a given namespace (i.e., the keys of the dictionary) It gives list of names comprising (some of) the attributes of the given object, and of attributes reachable from it.
vars( ) returns the full dictionary of identifier, value pairs

First-class objects are instances of a type that can be assigned to an identifier, passed as a parameter, or returned by a function.
Eg int and list, are clearly first-class types in Python.

scream = print() ***Python allows one function to be passed as a parameter to another.***
scream(Hello)

Modules and the Import Statement

from math **import** pi, sqrt	Allows direct use of the identifier, pi, or a call of the function, sqrt(2)
import math	Accessed using a fully-qualified name, such as math.pi or math.sqrt(2).

If there is file named utility.py, we could import that function from that module using the syntax:

from utility **import** count

The condition **if __name__ == '__main__'** is used in a Python program to execute the code inside the if statement only when the program is executed directly by the Python interpreter. When the code in the file is imported as a module the code inside the if statement is not executed

Existing Modules:

array	Provides compact array storage for primitive types.
collections	Defines additional data structures and abstract base classes involving collections of objects.
copy	Defines general functions for making copies of objects.
heapq	Provides heap-based priority queue functions
math	Defines common mathematical constants and functions.
os	Provides support for interactions with the operating system.
random	Provides random number generation.
re	Provides support for processing regular expressions.
sys	Provides additional level of interaction with the Python interpreter.
time	Provides support for measuring time, or delaying a program.

Pseudo-Random Number Generation

next = (a*current + b) % n; Generate random no each time you pass the argument

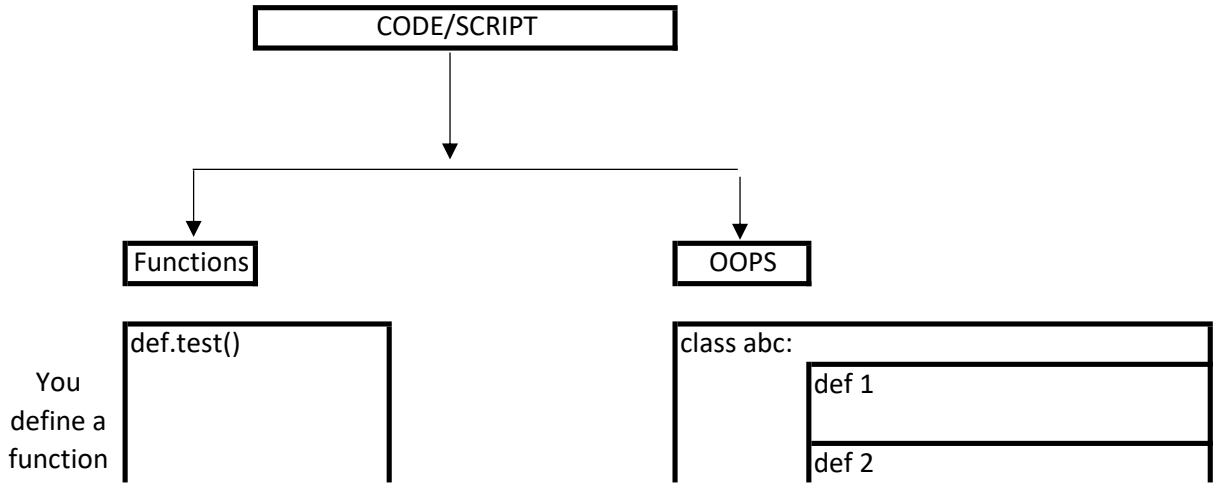
Syntax	Description
seed(hashable)	Initializes the pseudo-random number generator based upon the hash value of the parameter
random()	Returns a pseudo-random floating-point value in the interval [0.0,1.0).
randint(a,b)	Returns a pseudo-random integer in the closed interval [a,b].
randrange(start, stop, step)	Returns a pseudo-random integer in the standard Python range indicated by the parameters.
choice(seq)	Returns an element of the given sequence chosen pseudo-randomly.
shuffle(seq)	Reorders the elements of the given sequence pseudo-randomly.

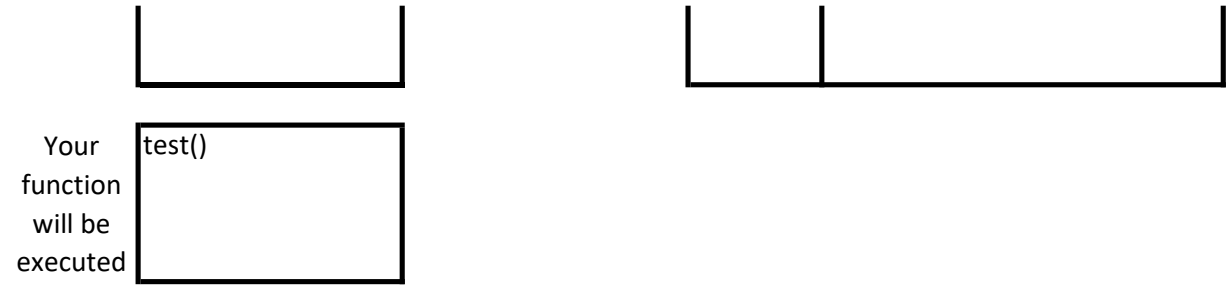
Classes and Objects

Creating a Class:
The first step in creating a class is giving it a name.

Instances of a Class: Objects and Attributes
An instance of an object is the realisation of a class,

Methods
Methods give you a way to change or interact with the object;
They are functions that interact with objects.

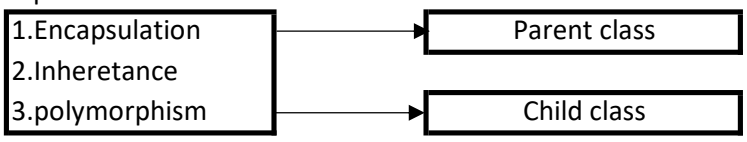


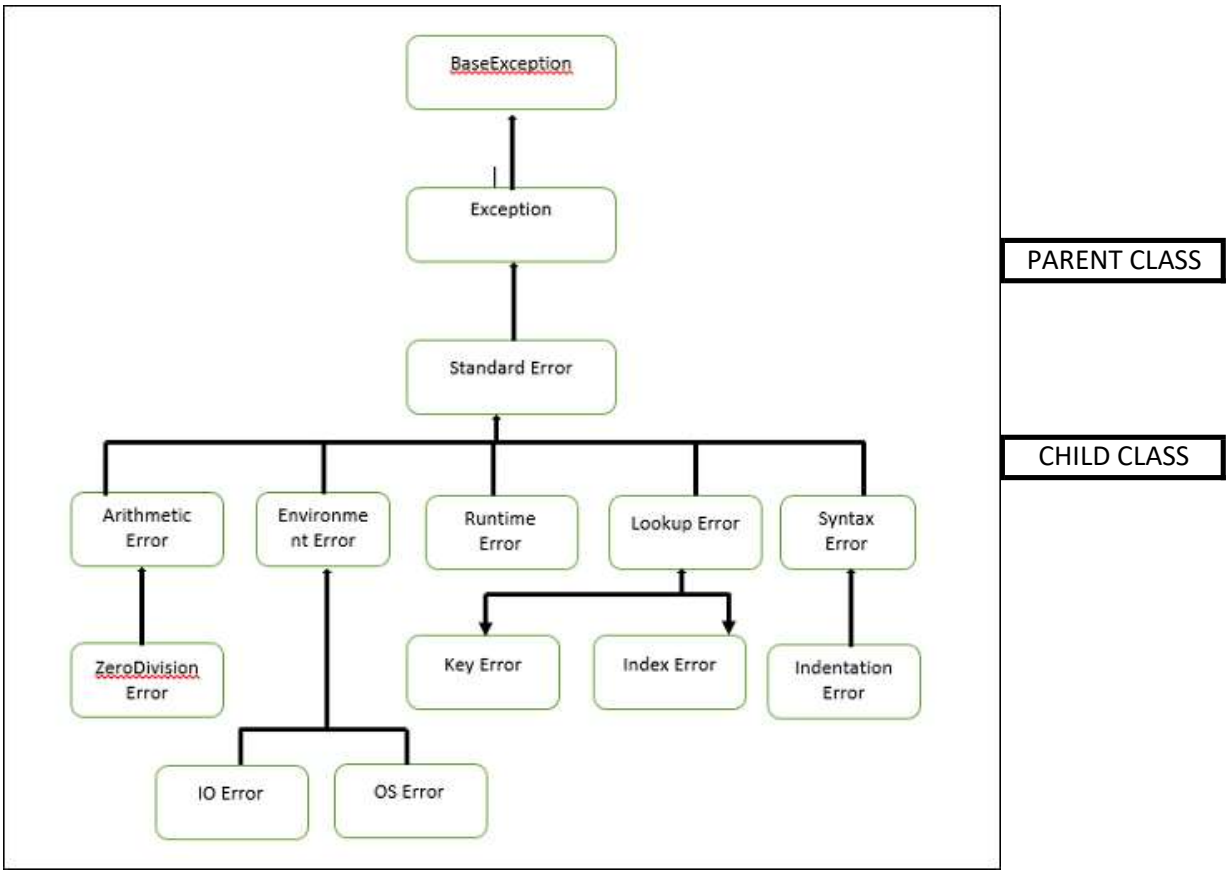


Your function will be executed

obj,abc()

3 pillars





Exception handling cannot rectify syntax error as it is checked by the compiler and handling can be done only at interpreter level.

Note: While mentioning class always make sure that you define child class before parent class

In case if it is mentioned vice versa then child class exception will not be executed and only parent class exception will run.

Note: You can also use traceback.format_exc() in place of sys.exc_info() and both the commands are used for debugging the entire project

EXCEPTION HANDLING			
Sr.No.	**Syntax/command**	**Output**	**Remarks**

1	a = int(input("enter first number"))
	b = int(input("enter second number"))
	try:
	  c = a/b
	  print(c)
	except:
	  print("b should not be zero")
		enter first number10
		enter second number0
		b should not be zero
			It will give except-ion case output and will not give an error for the same.

2	a = int(input("enter first number"))
	b = int(input("enter second number"))
	try:
	  c = a/b
	  print(c)
	except ZeroDivisionError :
	  print("b should not be zero")
		enter first number10
		enter second number0
		b should not be zero
			You can also mention the type of error after the except keyword. Parent keyword exception can also be used to address all types of errors.

3	try:
	  c=a/b
	  print(c)
	  d=a+b
	  print(d)
	
	except ZeroDivisionError :
	  print("b should not be zero")
	except ValueError:
	  print('please input number only')
			You can use multiple exception blocks for single try block

4	while True: print("sunny")		It will print the word infinitly until and unless you give break statement.

5	while True:
	  name = input("enter your name: ")
	  if name == "sunny":
	  print("sunny")
	  break
			Until the input given does not match the name specified, it will pop up again and again. Once matched the name will be printed just once as break is mentioned.

6	while True:
	  try:
	  a=int(input("enter first number"))
	  b=int(input('enter second number'))
	  d=a/b
	  print(d)
	  except ValueError :
	  print("there should not be string")
	  break
	  except ZeroDivisionError:
	  print("please enter non-zero denominator")
	  break
			The complete expcetion handling example

7	try:
	  a = int(input("enter a number"))
	  print(a)
	except:
	  print('string not allowed')
			To allow input of only numbers and not givie error in other cases.

8	while True:
	try:
	a=int(input("first number"))
	b=int(input('second number'))
	c=a/b
	print("div:" , c)
	break
	except:
	print("b should not be zero")
		first number10
		second number0
		b should not be zero
		first number10
		second number1
		div: 10.0
			The while true loop will run again and again unless you don't give proper values for input command. Also the loop will break only when correct input and output is given.

9	while True:
	try:
	a=int(input("first number"))
	b=int(input('second number'))
	c=a/b
	print("div:" , c)
	break
	except ZeroDivisionError as e:
	print€
		first number10
		second number0
		division by zero
			To use in-built comments for error as your own comments.

10	import sys
	while True:
	try:
	a=int(input("first number"))
	b=int(input('second number'))
	c=a/b
	print("div:" , c)
	break
	except:
	print(sys.exc_info())
		first number10
		second number0
		(<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero'), <traceback object at 0x0000022A4AE41140>)
			It will define in detail the exact error that is in the code. No need to mention anything else in the exception.

11	while True:
	try:
	a=int(input("first number"))
	b=int(input('second number'))
	c=a/b
	print("div:" , c)
	break
	except:
	a,b,c=sys.exc_info()
	print("Exception class" , a)
	print("Exception message" , b)
	print("Line number", c.tb_lineno)
		first number10
		second number0
		Exception class <class 'ZeroDivisionError'>
		Exception message division by zero
		Line number 6
			In order to show in a proper format the error msg from exc_info()

12	import traceback
	while True:
	try:
	a=int(input("first number"))
	b=int(input('second number'))
	c=a/b
	print("div:" , c)
	break
	except:
	print(traceback.format_exc())
			You can also use this in place of sys.exc_info() and both the commands are used for debugging the entire project

13	while True:
	  try:
	  a=int(input("first number"))
	  b=int(input('second number'))
	  if a<0 or b<0:
	  raise Exception("neg number not allowed")
	  c=a/b
	  print("div:" , c)
	  break
	  except ValueError:
	  print("please enter int only")
	  except ZeroDivisionError:
	  print("please enter non-zero int")
	  except Exception as e:
	  print(e)
			Raise exception is used to provide exception at a specific line of code and can also be used in try block.

14	class NegativeNumberException(Exception): pass		

15	while True:
	  try:
	  a=int(input("first number"))
	  b=int(input('second number'))
	  if a<0 or b<0:
	  raise NegativeNumberException("neg number not allowed")
	  c=a/b
	  print("div:" , c)
	  break
	  except ValueError:
	  print("please enter int only")
	  except ZeroDivisionError:
	  print("please enter non-zero int")
	  except NegativeNumberException as e:
	  print(e)
			When you want to use custom exception in place of pre-defined once

16	while True:
	  try:
	  a=int(input("first number"))
	  b=int(input('second number'))
	  c=a/b
	  print(c)
	  except:
	  print("non-zero denominator")
	  finally:
	  print("hello")
	  print("python")
			The finally exception command is used to keep important code execution in last part and to keep the data safe. Case 1: We have created one database but we forgot to close the connection. Case 2: On cloud service we forgot to terminate.

17	while True:
	  try:
	  a=int(input("first number"))
	  b=int(input('second number'))
	  c=a/b
	  print(c)
	  except:
	  print("non-zero denominator")
	  else:
	  print("hello")
			Else exception shall run on correct input output combination or else exception command will run.

18	def askint1():
	  while True:
	  try:
	  val = int(input("please enter int"))
	  break
	  except:
	  print("looks like you have nor entered a int")
	  continue
			While true will run the code in continuous loop unless you enter a integer.

19	def sqrt(x):
	if not isinstance(x, (int, float)):
	raise TypeError(x must be numeric)
	elif x < 0:
	raise ValueError(x cannot be negative
			Checking the type of an object can be performed at run-time using the built-in function, **isinstance**. In simplest form, isinstance(obj, cls) returns True if object, obj, is an instance of class, cls,

20	age = −1 # an initially invalid choice
	while age <= 0:
	try:
	age = int(input(Enter your age in years:))
	if age <= 0:
	print(Your age must be positive)
	except (ValueError, EOFError):
	print(Invalid response
			We use the tuple, (ValueError, EOFError), to designate the types of errors that we wish to catch with the except-clause.

21	except (ValueError, EOFError): pass		If we preferred to have the while loop continue without printing the Invalid response message. It quietly catches the exception, thereby allowing the surrounding while loop to continue.
22	factors(100)		Generates the series 1,100,2,50,4,25,5,20,10 but not generated in increasing order
23	divmod(a, b),		Returns the pair of values (a // b, a % b)
24	quotient, remainder = divmod(a, b)		
25	for x, y in [(7, 2), (5, 8), (6, 4)]:		there will be three iterations of the loop. During the first pass, x=7 and y=2, and so on.

26	def fibonacci( ):
	a, b = 0, 1
	while True:
	yield a
	a, b = b, a+b
			Fibonacci series

27	import math a = math.sqrt(3) print(a)		When you just import modules

28	from math import sqrt a = sqrt(3) print(a)		When you import modules with defination

29	class Circle(object):
	
	# Constructor
	def __init__(self, radius=3, color='blue'):
	self.radius = radius
	self.color = color
	
	# Method
	def add_radius(self, r):
	self.radius = self.radius + r
	return(self.radius)
	
	# Method
	def drawCircle(self):
	plt.gca().add_patch(plt.Circle((0, 0), radius=self.radius, fc=self.color))
	plt.axis('scaled')
	plt.show()
			Defining a class and creating an instance of the class

30	RedCircle = Circle(10, 'red')		
31	dir(RedCircle)		Give data attributes of the object
32	RedCircle.radius		Return radius
33	RedCircle.color		Return colour
34	RedCircle.drawCircle()		Show a circle as defined

OOPS - Object oriented programming

Class Classification of real world entity
Object Represents the real world entity

Constructor entity/function/default method by which you can give data/information to class.

Advantages of OOPs

Clarity of code.
Proper segmentation of the code
Reusability of the code
Cleanness inside code
Structure inside code

Object-Oriented Design Principles

Modularity
Modularity refers to an organizing principle in which different components of a software system are divided into separate functional units.

Abstraction
The notion of abstraction is to distill a complicated system down to its most fundamental parts.

Encapsulation
Encapsulation refers to using software system that not reveal the internal details of their respective implementations Eg. Single underscore character (e.g., secret) are assumed to be nonpublic

Software Development

Traditional software development involves several phases. Three major steps are:
1. Design
2. Implementation
3. Testing and Debugging

Design

Algorithms in a way that is intended for human eyes only are called **pseudo-code**.

Class diagram.
A standard approach to explain and document the design using class diagrams to express the organization of a program

Class:	CreditCard	
Fields:	customer bank account	balance limit
Behaviors:	get customer( ) get bank() get account() make payment(amount)	get balance() get limit( ) charge(price

Coding Style and Documentation

The main principles that we adopt are as follows:

• Python code blocks are typically indented by 4 spaces. It is strongly recommended that tabs be avoided, as tabs are displayed with differing widths across systems, and tabs and spaces are not viewed as identical by the Python interpreter

• Use meaningful names for identifiers

Classes (other than Python's built-in classes) should have a name that serves as a singular noun, and should be capitalized (e.g., CreditCard).

Functions, including member functions of a class, should be lowercase. be separated by underscores. (e.g., makepayment)

Names that identify an individual **object** (e.g., a parameter, instance variable, or local variable) should be a lowercase noun (e.g., price).

Identifiers that represent a value considered to be a **constant** are traditionally identified using all capital letters and with underscores to separate words (e.g., MAX SIZE).

• Use comments that add meaning to a program and explain ambiguous or confusing constructs

Documentation

Python provides integrated support for embedding formal documentation directly in source code using a mechanism known as a **docstring.**
By convention, those string literals should be delimited within triple quotes (""").

Testing and Debugging

Testing is the process of experimentally checking the correctness of a program

Debugging is the process of tracking the execution of a program and discovering the errors in it

1. Testing

There are two main testing strategies,
top-down and bottom-up,

Top-down testing proceeds from the top to the bottom of the program hierarchy

if name == __main__ : *# perform tests...*
to test the functionality of the functions and classes specifically defined in that module.

More robust support for automation of unit testing is provided by Python's **unittest module**.
This framework allows the grouping of individual test cases into larger test suites, and provides support for executing those suites, and reporting or analyzing the results of those tests using **regression testing**

2. Debugging

The simplest debugging technique consists of using **print statements** to track the values of variables during the execution of the program.

Other approach is using **debugger** The basic functionality provided by a debugger is the insertion of **breakpoints** within the code

Inheritance

Inheritance allows a new class to be defined based upon an existing class as the starting point. In object-oriented terminology, the existing class is typically described as the base class, parent class, or superclass, while the newly defined class is known as the subclass or child class.

Protected Members

Protected or Private access modes.
Members that are declared as **protected** are accessible to subclasses, but not to the general public, while members that are declared as **private** are not accessible to either.

Abstract Base Classes

A class is called abstract base class if its only purpose is to serve as a base class through inheritance.
Eg. Progression class, which serves as a base class for three distinct subclasses: ArithmeticProgression, GeometricProgression,and FibonacciProgression

Nested Classes

```
class A: # the outer class
  class B: # the nested class
```

nest one class definition within the scope of another class.

Dot operator

Python interpreter begins a name resolution process, described as follows:
1. The instance namespace is searched; if the desired name is found, its associated value is used.
2. Otherwise the class namespace, for the class to which the instance belongs,is searched; if the name is found, its associated value is used.
3. If the name was not found in the immediate class namespace, the search continues upward through namespace the inheritance hierarchy, checking the class for each ancestor (commonly by checking the superclass class, then its superclass class, and so on). The first time the name is found, its associate value is used.
4. If the name has still not been found, an AttributeError is raised

Inheritance	Multiple	class bank:
		class HDFC_bank:
		class icici(bank , HDFC_bank):
		  pass
			In case both hdfc_bank() and bank() class have the same defined function then that function will be executed based on the order mentioned in icici() i.e function which is mentioned first in the order will be executed.

	Multi-layer	class bank :
		class HDFC_bank(bank):
		class icici(HDFC_bank):
		  pass
			when same function is mentioned in both bank() and HDFC_bank(bank) then function from HDFC_bank(bank) will overrite the function from bank().

	class objects modules		

packages
constructor
inheritance
private public protected
abstraction
encapsulation
polymorphism

OOPS concept			
Sr.No.	**Syntax/command**	**Output**	**Remarks**
1	__init__(self):		Initilization of a particular variable to class
2	self.name = name		Public variable
3	self._surname = surname		Protected variable (Encapusulation)
4	self.__yob = yob		Private variable (Encapusulation)
5	print(adit._person__name)		You can only access this private variable by appending the class name before the variable
6	import test1		For accessing code from one python file into another
7	from test2 import person		For importing specific class from python file into another
8	from utils.util import person2		For accessing class from a module(py file) inside package(folder)
9	__students = "data science"		This is called **data abstractiion** as we are trying to hide that data behind the local variable. It is the process of hiding the real implementation of an application from the user and emphasizing only on usage of it

10	class ineuron:
	  __students = "data science"
	
	  def students(self):
	  print("print the class of students",ineuron.__students)
	
	i = ineuron()
	i.students()
	print(i._ineuron__students)
			Such data can be accessed only by appending class name before it with single underscore.

	class car :
	
	  def __init__(self, body, engine, tyre):
	  self.body = body
	  self.engine = engine
	  self.tyre = tyre
	
	  def mileage(self):
	  print("Mileage of this car")
	
	class tata(car):
	  pass
			When u want to utilize entire code from one class (parent class) into another class (child class) it Is called **Inheritance.** Here tata class will exactly inherit car class

	class bank :
	
	  def transaction(self):
	  print("Total transaction value ")
	  def account_opening(self):
	  print("This will show you your account opening status")
	  def deposit(self):
	  print("This will show you your deposited amount")
	
	class HDFC_bank(bank):
	
	  def HDFC_to_icici(self):
	  print("This will show all the transactions happend to icici from HDFC")
	
	class icici(HDFC_bank):
	  pass
			Multi level inheritance

	class bank:
	
	  def transaction(self):
	  print("Total transaction value ")
	  def account_opening(self):
	  print("This will show you your account opening status")
	  def deposit(self):
	  print("This will show you your deposited amount")
	  def test(self):
	  print("This is test method from bank")
	
	class HDFC_bank:
	
	  def HDFC_to_icici(self):
	  print("This will show all the transactions happend to icici from HDFC")
	  def test(self):
	  print("This is test method from HDFC bank")
	
	class ineuron_bank:
	
	  def account_status_icici(self):
	  print("Print a account status in icici")
	
	class icici(bank , HDFC_bank, ineuron_bank):
	  pass
			Multple inheritance

11	class ineuron:
	  def __init__(self):
	  self.students1 = "data science"
		data analytics	In run time you can overwrite the value of the variable

	i = ineuron() i.students1 = "data analytics" print(i.students1)		

12	class ineuron1:
	  def __init__(self):
	  self.__students1 = "data science"
			In run time you cannot change the value of variable that is private but can only be changed if a function is

	i1._ineuron__students1 = "data analystics" i1.students1()		defined to reassign the value. This is called as **encapsulation.** It puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data

	def students_change(self):
	self.__students1 = "Big data"
	def students_change_to(self, new_value):
	self.__students1 = new_value		

13	class ineuron:
	
	def students(self):
	print("Student details")
	
	class class_type:
	
	def students(self):
	print("print the class type of students")
			Polymorphism means multiple forms. It is the property where a single function performs in varied different ways based on the values.

	def ineuron_external(a): a.students()		
	i = ineuron() j = class_type() ineuron_external(i) ineuron_external(j)		
14	a is b		Evaluate whether identifiers a and b are aliases for the same object
	a == b		whether the two identifiers reference equivalent values
14	super(). init (customer, bank, acnt, limit)		Calls the init method that was inherited from the parent class or superclass and is initiated in child class or subclass
15	palette = warmtones		Both alias and subsequently add or remove colors from “palette,” we modify the list identified as warmtones.
16	palette = list(warmtones)		This will create a ***shallow copy*** of warmtones
17	palette = copy.deepcopy(warmtones)		In ***deep copy*** new copy references its own copies of those objects referenced by the original version.