Python - Introduction:

Aug 02, Tue

IMPORTANT about PYTHON:------

Python is case sensitive. Files extension: filename.py

We can take single, double, triple quotes for print statements.

Semi colon (:) is must after if and esle like conditions.

Indentation is more important in python.

****

Programming:

A way to instruct the computer to perform various tasks.

Eg: only *PLANNING* to go to friend marriage

Coding:

Process of transforming ideas, solutions and instructions into the language that computers understand (binary / machine code).

Eg: *IMPLEMENTATION* of planned program to go to friend marriage

** Computer can understand only binary language (0 and 1) **

Character	Binary Code	Character	Binary Code	Character	Binary Code	Character	Binary Code	Character	Binary Code
A	01000001	Q	01010001	g	01100111	w	01110111	-	00101101
B	01000010	R	01010010	h	01101000	x	01111000	.	00101110
C	01000011	S	01010011	i	01101001	y	01111001	/	00101111
D	01000100	T	01010100	j	01101010	z	01111010	0	00110000
E	01000101	U	01010101	k	01101011	!	00100001	1	00110001
F	01000110	V	01010110	l	01101100	"	00100010	2	00110010
G	01000111	W	01010111	m	01101101	#	00100011	3	00110011
H	01001000	X	01011000	n	01101110	$	00100100	4	00110100
I	01001001	Y	01011001	o	01101111	%	00100101	5	00110101
J	01001010	Z	01011010	p	01110000	&	00100110	6	00110110
K	01001011	a	01100001	q	01110001	'	00100111	7	00110111
L	01001100	b	01100010	r	01110010	(	00101000	8	00111000
M	01001101	c	01100011	s	01110011	)	00101001	9	00111001
N	01001110	d	01100100	t	01110100	*	00101010	?	00111111
O	01001111	e	01100101	u	01110101	+	00101011	@	01000000
P	01010000	f	01100110	v	01110110	,	00101100	_	01011111

Compiler:

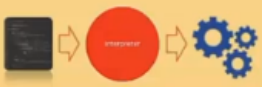
Software that converts program written in high-level language (source

language) to low-level language (object/Target/Machine language).

Interpreter:

Translator to convert programs from high-level language to low-level. It translates line by line and reports the error during the process. But Compiler will translate all at a time and finally reports error if any.

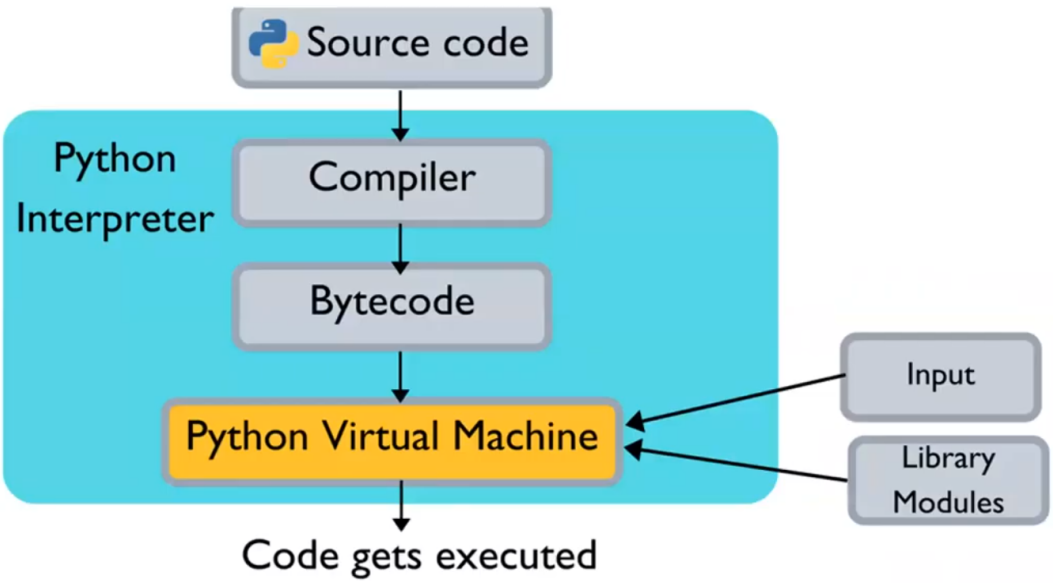
Compiler vs Interpreter:

Compiler	Interpreter
Converts the whole high level language program to machine language **at a time**	Converts the whole high level language program to machine **line by line**
Compilation is **slow**	Translation is done **fast** (line by line)
Execution is **fast**	Execution is **slow**
Generates an intermediate code (executable file)	Execute the program **without any intermediate code**
Errors are displayed **after the entire program is compiled**	Errors are displayed **for each single instruction**
	

Python Execution:

Source code ---> Python Byte Code ---> PVM ---> Output
(basic.py) (basic.pyc) (Python Virtual Machine)

PYC files have set of instructions to follow in sequence to generate output. These are faster than normal python code files.



Python is both compiled and interpreted language.

Python is high-level. interpreted. interactive and object-oriented scripting language.

Python History:

Developed by Guido van Rossum in 1991 at Netherlands before java. Drived from many languages like ABC. Modula-3. C. C++. Algo-68. SmallTalk .Unix shell and other scripting languages.

Python Features:

Why Python Popular:

1.Python has large user community (9M).
2.It has well-mainted libraries. (approx 13k+)
3.Online guidance (StackOverflow)
4.Second most using language in the world after javascript
5.Python most popular for Data Science. ML IoT apps and least popular for Mobile and Web apps.

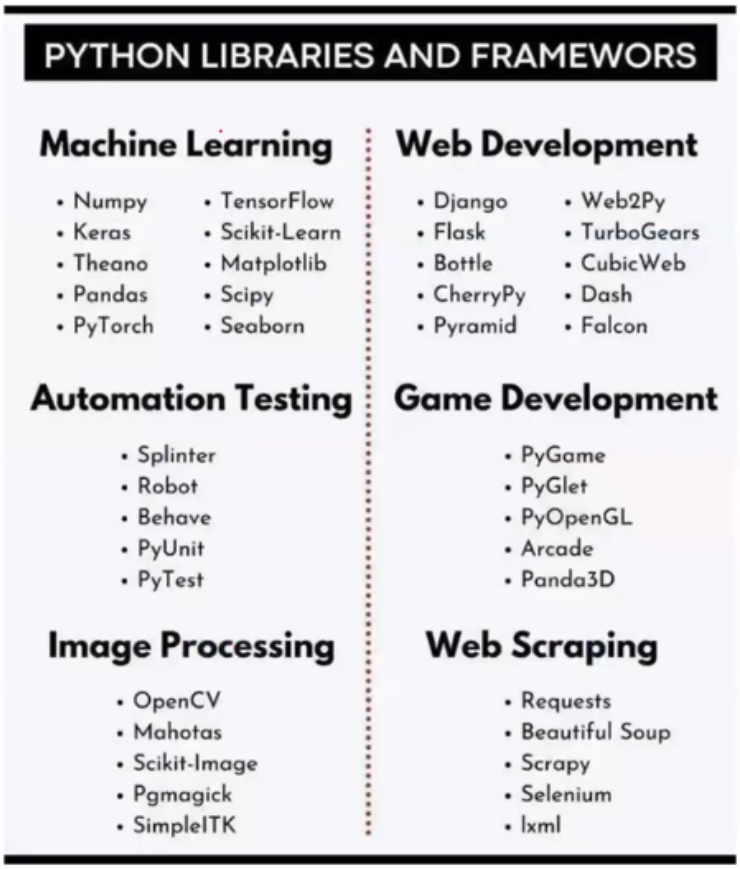
Module vs Package vs Library vs FrameWork:

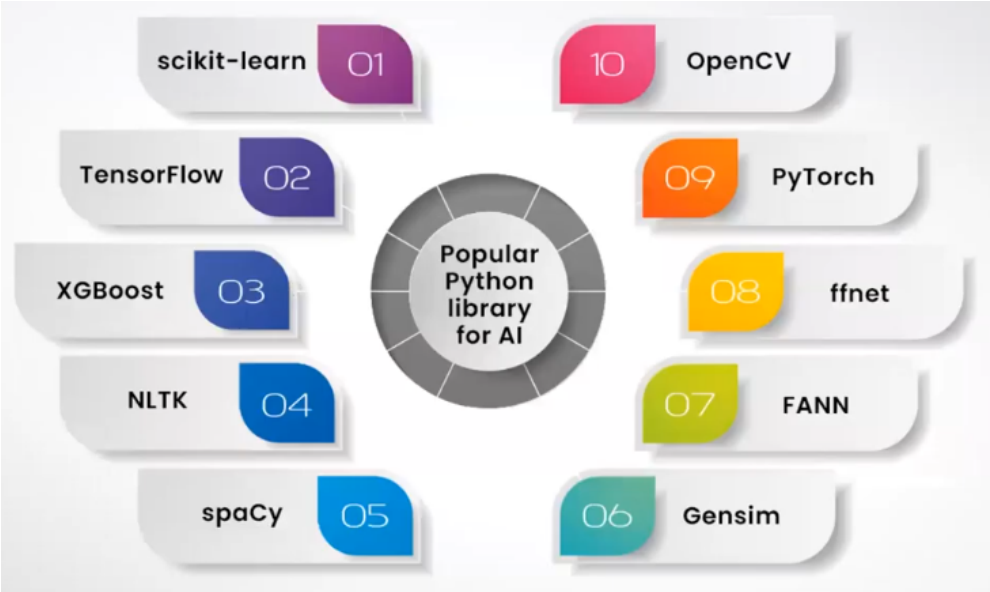
Module: Simple Python file contains collection of functions and global variables having .py extension file.

Package: Collection of Modules

Library: Collection of Packages.

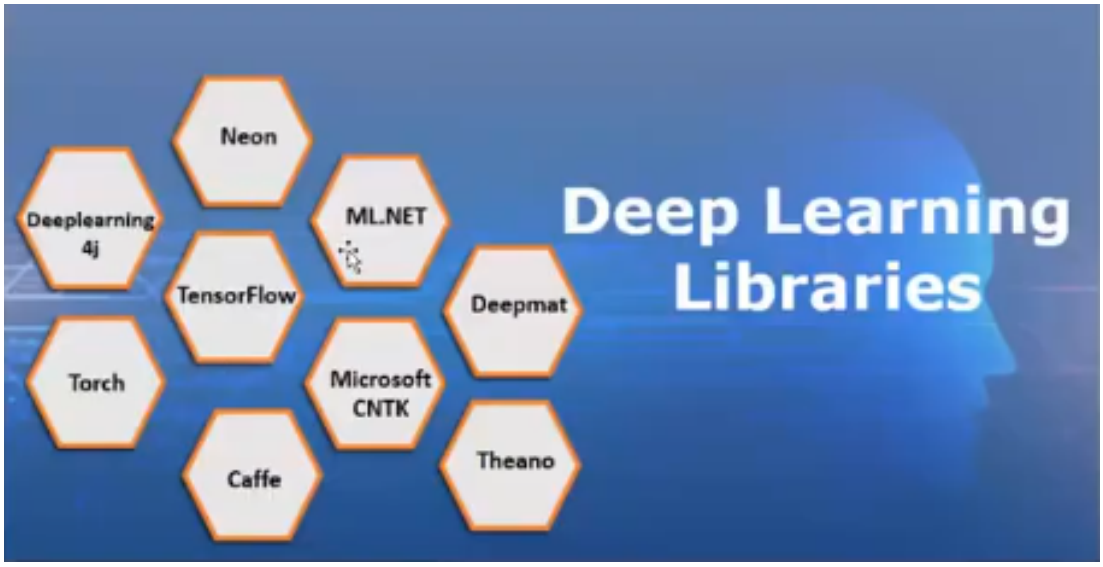
Eg: Pandas. Numpy. Matplotlib. Seaborn.



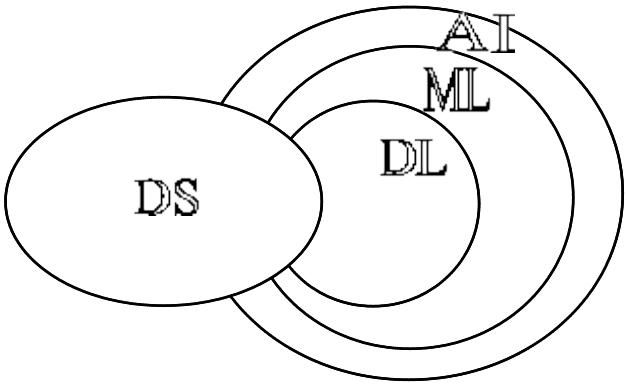


Framework: Collection of Modules or Packages or Libraries and with architecture MVC (Model View Controller) or MVT (Model View Template)

* Must and should learn any library/framework in interested domain along with Python to get a job.


Neon
Deeplearning 4j
ML.NET
TensorFlow
Deepmat
Torch
Microsoft CNTK
Caffe
Theano
Deep Learning Libraries


django
pyramid
Falcon
Flask
Bottle
Python Web Frameworks
WEB2PY
FastAPI
Tornado
SANIC
Dash
by plotly


AI
ML
DL
DS

Flavours of Python:

Python clubbed with many other technologies and forms various flavours.

1.Jython - Java + Python
2.IronPython - .Net + Python
3.RubyPython - Ruby +Python
4.Cython - C+Python
5.Stackless Python - Experimental flavour
6.Python 3 or 3K

IDEs:

PyCharm. Spyder. PyDev. Idle. Wing. Eric Python. Rodeo. Thonny

Source Code Editors:

Sublime Text. Atom. Vim. **Visual Studio Code**

Job Roles for Only Python:

1. Python Developer
2. Django Developer

Job Roles for Data Science:

1. Data Analyst
2. Data Engineers
3. Database Administrator
4. Machine Learning Engineer
5. Data Scientist
6. Data Architech
7. Statistician
8. Business Analyst
9. Data and Analytics Manager
10. Deep Learning Engineer.

Python - Variables & Datatypes — Aug 03. Wed

Comments:

Comments enhance the readability of code and help programmers understand code easily.

Comments are ignored by the interpreter during the execution of code.

Comments are three types:

1. Single Line Comment
2. Multi Line Comment
3. Doc String

To put single line comment use "#" before line.
Use ''' ''' (triple quotes) for multi line comments.

Variables:

Variables are containers having data values or Name of the memory location is called variable.

Eg: Name = John. Age = 35 (Here Name & Age are variables. John & 35 are values)

System automatically assign memory locations to data using heap.

Values can be assign to variables in 3 types:

Name = John (Single value assign to single variable)
x=y=z=50 (Single value assign to multiple variables)
x,y,z = 1,2,3 (Multiple values assign to multiple variables)

If many values assigned to same variable, recent value will be dispaly in output. Because python will process line by line

Eg:
```
x=10
x=20
print(x) (This will give 20 as output)
```

Should not use quotations in paranthesis while calling variables.

id() will use to find out Memory location of a variable

Eg: kumar=1222
print(id(kumar)) # (Output: 2280036228784)

Same data will have same memory location to mitigate duplication.

Rules for Variablename:

1. Must start with letter or underscore
2. Cannot start with number
3. Can only contain alpha-numeric (A-z, 0-9 and _)
4. Case sensitive (Name NAME name - all three are different)
5. Reserved keywords cannot use

Identifiers:

They are user-defined names, used to specify the names of variables, functions, classes, modules and objects.

Keywords:

35 Keywords

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
'while', 'with', 'yield']

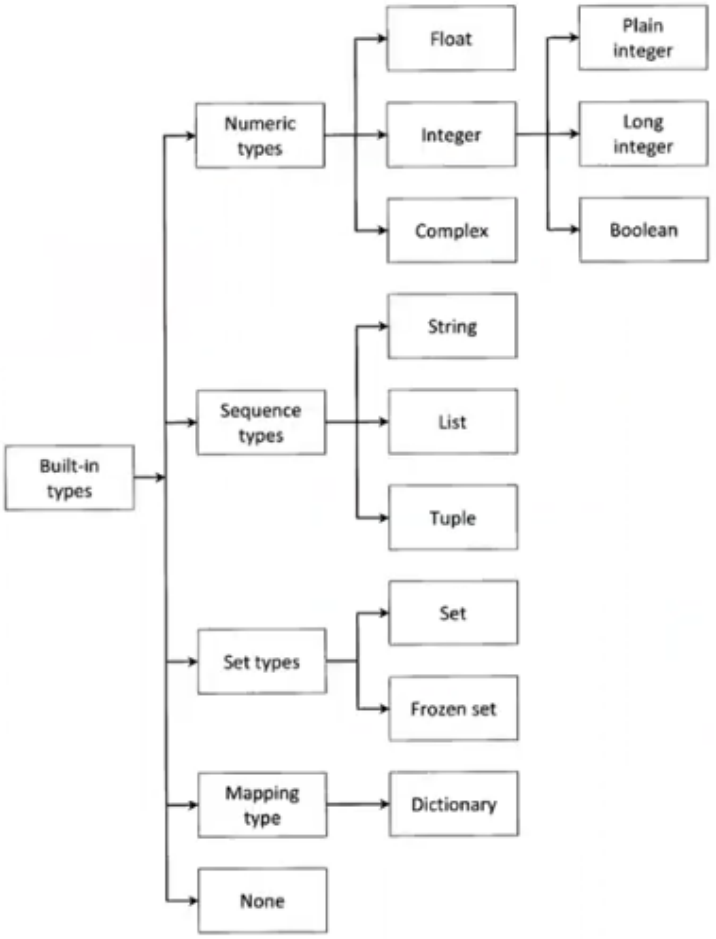
They are reserved words for special meaning and are used for special purpose. We can't use them for variable, class, function, module and object.

Cases usage:

camelCase : Starting small case and Middle big case
snake_case : All are small case with underscore
PascalCase : Starting and Middle are big case.

Data Types:

It's nothing but type of a value holded in variable.



** Plain Integer. Long Integer are deprecated in latest version.

Whole numbers (-inf to 0 to inf) are 'int' data type
Decimals are 'float' type
Real + Imaginary are 'complex' type
True and False are 'bool' type. True means 1 and False refer 0.

To know type of data use 'type()' function. One argument allowed at a time.

Type Conversion:

Conversion from one data type to another.

Two types of conversions are there in python:

1. Implicit Conversion
2. Explicit Conversion

Implicit Type Conversion:

Python interpreter **automatically** converts one data type to another.

small datatype (**int**) ---------- No Data Loss ----------> Big datatype (**float**)

Datatype built-in functions:

1. int()
2. float()
3. bool()
4. str()

Explicit Type Conversion:

Python interpreter **forcefully** converts one data type to another.

Big datatype (**float**) ---------- Data Loss ----------> small datatype (**int**)

Dynamic data passing:

Rather than assigning a value to variable every time. we can pass dynamically using 'input()' function.

```
Eg: d=10                                        #static
    name=input("Enter the name:")               #input function
    print(name)
```

'input' function will ask user input at run time from terminal/console.

'input()' by default returns string type but we can alter it.

```
Eg: d=10                                        #int
    name=int(input("Enter the no:"))      #Only integers only allowed
```

```
print(type(name))
```

Eg2:

```
name=input("Enter the name:")
roll=int(input("Enter the Roll no: "))
M_status=bool(int(input("Enter the status")))
print(name,roll,M_status)

Output# :   kiran    1    True
```

When bool take directly, it will return 'True' while we give either 1 or 0. We need to do type conversion to get correct result as above.

****

Task - 4:

Practice comments, datatypes, variables

Task - 5:

Compound interest, Simple interest

Task - 6:

Prepare a notes on today topic

Python - Operators & Control Statements Aug 04. Thur

Operators:

The operator will perform operations on operand.

Eg: a + b (here a and b are operand. + is operator)

Types of Operators:

1. Arithmetic
2. Logical
3. Membership
4. Assignment
5. comparision
6. identity
7. bitwise

Arithmetic Operators: + - * / % ** //

+, -, *, /, %.
** (exponent like 5**2=5 square).
// (Integer division or floor division 5/2=2 -5/2=-3)

Assignment Operators: = += -= *= /= %= **= //=

=, +=, -=, *=, /=, %=, **=, //=

Logical Operators:

True=1, False=0

A	B	A and B	A	B	A or B	A	Not A
True	True	True	True	True	True	True	False
True	False	False	True	False	True	False	True
False	True	False	False	True	True		
False	False	False	False	False	False		

Comparision Operators: < > <= >= == !=

<, >, <=, >=, ==, !=

Control Statements:

Control statements are 3 types:

1. Conditional statements/ Decision Making Statements

   Eg: if, if-else, if-elif-else, nested if-else

2. Transfer Statements / Jumping Statements

   Eg: break, continue, pass

3. Iterative Statements / Looping Statements

   Eg: for, while

Conditional Statements:

short hand if statements are introduced newly in python.

1. if the statement
2. if else statement
3. nested if statement
4. if.. elif ladder
5. short hand if statement
6. short hand if-else statement

if-else flow:

```
if condition:
        statements
else:
        statements
```

Eg:
```
if grade>=70:
    print("pass")
else:
    print('fail')
```

if-elif-else flow:

```
if condition:
        statements
        statements
        statements
elif condition:
        statements
        statements
        statements
else:
        statements
        statements
        statements
```

if we want to check more than one condition, then will use elif.
we can use either elif block or 'and'

short hand if:

if condition written in single line

Eg: ram=20
syam=30
if ram<syam:print('this is if')

short hand if-else:

If condition true, LHS of if will execute. if condition false, RHS of if will execute.

Eg2: print('this is if') if ram<syam else print('this is else')

Looping / Iterative Statements:

A program loop is a series of statements that executes for a specific number of repititions or until specified conditions are met.

1. For loop
2. While loop

Differences between 'for' and 'while' loops:

We can write range() function in 'for' but not in 'while' loops
We can loop every element in 'for' but not in 'while' loops
We can infine loop in 'while' but not in 'for' loop

** do-while not there in python **

While loop:

```
syntax: while condition:          # until condition false, execution will loop
            statement
            statement
        else:
            statement
            statement
```

For loop:

For will check everything but while only check if condition is true

```
for var in iterable:
        statement
        statement
else:
        statement
        statement
```

syntax: for var in iterable:

for=keyword.
var=temporary variable.
in=membership operator.
iterable=list/dictionary/string/tuple/range function

range function:

It consists of Start, Stop, Step (SSS). range(Start, Stop, Step)

```
Eg: range(0,11,2)          # this will print 0 to 10 with 2 nos gap
```

Transfer Statements:

break:

its used to break the for loop

```
Eg: for i in 'kiran':
        if i=='r':
            break
        else:
            print(i)
```

Continue:

Using continue will skip the current iteration

Eg: for i in 'kiran':
    if i=='r':
        continue
    else:
        print(i)

pass:

pass is just used to skip the current execution without generating error.

Eg: if True: # this will throw error

Eg2: if True:
    pass # this will just ignored by interpreter.

Tables using nexted for loop:

```
for x in range(1,100):
    for y in range(1,11):
        print(x*y,end=' ')
    print()
```

****

Task - 7:

Practice total control statements

Task - 8:

Explore nested while

Task - 9:

Explore patterns using for loop

Python - List & List methods — Aug 05. Fri

List:

1. Collection of values or items of different data types separated with comma and senclosed with square brackets [ ].
2. The element of the list can access by index starting from 0 ending (n-1)
3. Lists are Mutable (Can modify) types.

Eg: hari=[1. 3.33. True. 'kiran'] # 1 - index 0: 'kiran' - index 4

Array elements have same data type but index can have different type

Forward/backward indexing:

As indexing moving right side is forward/positive indexing

As indexing moving left side is backward/negative indexing

[1. 2.2. 3. 4. 9. 10]

index is respectively 0. 1. 2. 3. 4. 5 from left to right

index is respectively -6. -5. -4. -3. -2. -1 from right to left

Eg: hari=[1. 3.33. True. 'kiran']
print(hari[3]) # Output: 'kiran'

List Characterstics:

1. Ordered:	Maintain the order of the data insertion.
2. Changeable:	List is mutable and we can modify items.
3. Heterogeneous:	List can contain data of different types.
4. Contains duplicate:	Allows duplicates data

Slicing:

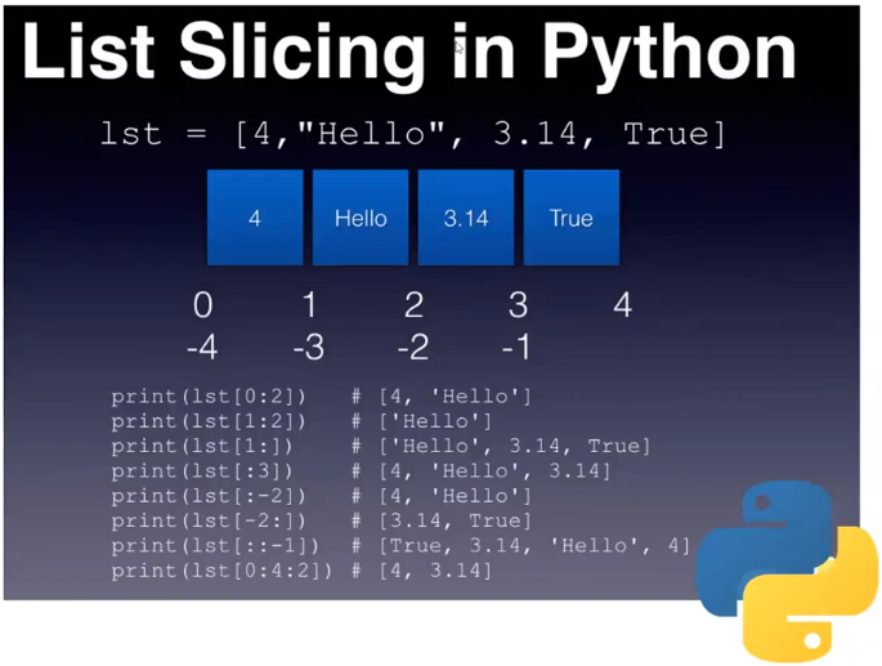
Nothing but cutting the list items as per our requirement

Slicing will follow SSS pattern. [Start:Stop:**Skip**]

Skip also will follow same n-1 rule like index. i.e. Requirement to **skip** 2 values means. take skip=3.

If **skip**=1. no skipping happends here and give same result. skip=-1. will return empty list.
If need to give - values to skip. start and stop should be interchange. So that the index wil shift to backward index.

```
0:          # print all values right side 0 index
:2          # print all values left side 2 index
::          # print all values in list
::-1        # print all values in reverse
-1          # print last element in list
```



List methods/functions:

append. extend. copy. clear. count. index
insert. pop. remove. reverse. sort.

Variablename function declaration:

syntax: variablename.method()

append():

append will add passed value at the end of list.

Eg: akilesh=[1.2.3.4.5.6.7.7]
akilesh.append('kiran') # kiran added at end of list.
print(akilesh)

***extend*():**

If tried to add more than one element using append. it will treat both as single element and add. In extend. it will treat separate elements and add.

Eg: akilesh=[1.2.3.4.5.6.7.7]
akilesh.extend('kiran') # k. i. r. a. n added at end of list.
print(akilesh)

Eg2: akilesh=[1.2.3.4.5.6.7.7]
akileshl=[1.2.3.4.5.6.7.7]
akilesh2=[1.2.3.4.5.6.7.7]
akilesh.extend('kiran')
akilesh.append(['kiran'].['python'])
akilesh.extend(['kiran'].['python'])
print("extend".akilesh)
print("append".akileshl)
print("extend".akilesh2)

So. we need to take extend to add multiple elements to list.

***Copy*():**

Once copy() used. original will be converted into immutable and any changes will not affect to original.

Eg: akilesh=[1.2.3.4.5.6.7.7]
a=akilesh.copy()
akilesh.append('python')
print(a)

***Clear*():**

This is remove all the items in the list.

Eg: akilesh=[1.2.3.4.5.6.7.7]
akilesh.clear()
print(akilesh) # can use print(akilesh.clear())

***Count*():**

It will count the passed argement how many times repeated in list

Eg: akilesh=[1.2.2.2.3.4.5.6.7.7]
print(akilesh.count(2))

***len*():**

This will give total number of elements in the given list.

Eg: akilesh=[1.2.2.2.3.4.5.6.7.7]
print(len(akilesh)) # len is built-in function hence syntax different

***index*():**

This will result the index of the passed value. If duplicate values there in list. it will give first one index only. Index will tell the position of vaue. multiple elements can be view using for loop.

Eg: akilesh=[1.2.2.2.3.4.5.6.7.7]
print(akilesh.index(2)) # Output: 1 as it will give 1st no index

Eg2: strl=[1.2.2.2.3.4.5.6.7.7]
for i in range(len(strl)): # Print all index.
if strl[i]==2:
print(i)

***insert*():**

This will add input at required index position. We can add one value at one index position.

Eg: akilesh=[1.2.2.2.3.4.5.6.7.7]
akilesh.insert(0,'kiran')
print(akilesh)

***pop*():**

This will remove value of given index position. Pop will take index

Eg: ak=[1.2.2.2.3.4.5.6.7.7]
ak.pop(0)
print(ak)

***remove*():**

This will remove given element while pop will remove index. remove will delete 1st value of given element if duplicates are present. multiple elements can be deleted using for loop.

Eg: ak=[1.2.2.2.3.4.5.6.7.7]
ak.remove(2)
print(ak)

reverse():

This will reverse the order of list items. We can do the samething using slicing but its manaul which is time consuming. This is using function it will take less time.

Eg: ak=[1.2.2.2.3.4.5.6.7.7]
ak.reverse()
print(ak)

sort():

This will assign the list items in the ascending order.

Eg: ak=[4.5.7.8.4.7.3.7.3.7.1.0]
ak.sort() # ak.sort(reverse=True) for descend
print(ak)

list comprehension:

We can use for loop. if condition in single list.we use list in comprehension. This is like same as short hand if-else.

syntax: list=[expression if con for item in sequence]

Eg: list=["EVEN" if i % 2 == 0 else 'odd' for i in range(10)]
print(list)

Here 1-9 store in i initially. if i%2 ==0. LHS execute else RHS will execute.

****

Task 10:

Practice list and list methods

Task 11:

Write calculator program using python

Task 12:

Prepare word document on list topic

Tips for Task 11:

+ - * /

step1: take 2 operands
step2: ask user what operation
step3: show the result

print. if. elif. input. for. while - can be used.

ctrl+? - comments

Python - String & String Methods — Aug 08. Mon

Strings:

Collection of characters surrounded by single/double/triple quotations

Eg: hari='sai'/ "sai" / '"sai"'
print(type(hari))

Single quotes used to write normal text
While giving aposterphe. can use double quotes. Recommonded.
More than online and paragraphs will be written in triple quotes.

String = value in triple quotes assigned to variable.
Comment=anything just write in triple quotes without assign to variable..

String Methods:

slice	upper	lower	count	startswith
endswith	find	format	index	isalnum
isdigit	split	lstrip	rstrip	strip
title	removeprefix	removesuffix	replace	join

slice():

Nothing but cutting the list items as per our requirement

Slicing will follow SSS pattern. [Start:Stop:**Skip**]

Skip also will follow same n-1 rule like index. i.e. Requirement to **skip** 2values means. take skip=3.

If **skip**=1. no skipping happends here and give same result. skip=-1. willreturn empty list.

If need to give - values to skip. start and stop should be interchange. Sothat the index wil shift to backward index.

0:# print all values right side 0 index
:2# print all values left side 2 index
::# print all values in list
::-1# print all values in reverse
-1# print last element in list

Eg: a='this','is','my','book','select','your','wish'
print(a[2:6:2])

Eg2: String = 'ASTRING'
s1 = slice(3)
s2 = slice(1, 5, 2)
s3 = slice(-1, -12, -2)
print(String[s1])
print(String[s2])
print(String[s3])

upper():

convert given string into upper case, only if method assigned to a variable.

Eg: satish='My name is james bond'
o=satish.upper()
print(o)

If return value is string, must assign method to a variable. In case of return type is int no issue.

lower():

convert given string into lower case, only if method assigned to a variable.

Eg: satish='MY NAME IS JAMES BOND'
o=satish.upper()
print(o)

len():

Return length of the variable data

Eg: biryani="i love biryani"
print(len(biryani))

count():

Will return how many times the given string repeating

Eg: biryani="i like biryani biryani"
print(biryani.count('biryani'))

***startswith*():**

Will return True/False if the given variable starts with respective string

```
Eg: s="sampoornesh babu"
d="kiran babu"
print(s.startswith('sa'))
```

***endswith*():**

Will return True/False if the given variable ends with respective String

```
Eg: s="sampoornesh babu"
d="kiran babu"
print(s.endswith('bu'))

Eg2: websites=['abc.gov','xyz.in','dfg.in','asw.com']
in_websites=[]
        for k in websites:
                if k.endswith('in'):
                in_websites.append(k)
print(in_websites)
```

***find*():**

Will return the starting index of the given string in variable

```
Eg: ajay=" today is working"
print(ajay.find('is'))
print(ajay.index('is'))
```

find and index both will return samething but they differed in error cases.

```
Eg: ajay=" today is working"
print(ajay.find('kiran'))       # output: -1
print(ajay.index('kiran'))      # output: error
```

kiran is not in the variable. hence both find and index will throw error.

Using find error. we can simply write code as " if -1 returns there is no value"

***format*():**

string formatting is used to change the required part of string in variable in the same format.

Eg: d="hey {} inka work avaleda" # {} - **called as place holder.**
d.format('kiran')

Above can be written in a single line as:

Eg: print("hey {} inka work avaleda".format('kiran'))

We can use same format with various names:

Eg3: list_names=['siva','chandra','praveen','mukesh']
for j in list_names:
    v="{i} thinava neekosam 30per discount vundi".format(i=j)
print(v)

isalnum():

This will check whether the given variable value is alpha numeric or not and returns True/False.
If the variable value is integer. even it will show True because integer is also a nueric.

Eg: rathnakar="ll"
print(rathnakar.isalnum())

Eg: rathnakar="adi"
print(rathnakar.isalnum())

isdigit():

This will check whether the given variable value is digit or not and returns True/False.
If integers given it will show True. If alphanumeric/float values given. it will show False.

Eg: rathnakar="1234"
print(rathnakar.isdigit()) # output: True

Eg2: rathnakar="adil234"
print(rathnakar.isdigit()) # output: False

strip():

Removes spaces before and after variable value.

Eg: pramodh=" i want more space .... "
print(len(pramodh))
v=pramodh.strip()
print(v)
print(len(v))

lstrip():

Removes spaces left side of variable value.

Eg: pramodh=" i want more space .... "
print(len(pramodh))
v=pramodh.lstrip()
print(v)
print(len(v))

rstrip():

Removes spaces right side of variable value.

Eg: pramodh=" i want more space .... "
print(len(pramodh))
v=pramodh.rstrip()
print(v)
print(len(v))

split():

Converts given string as list

Eg: p="chaduvu mundu"
print(p.split())

title():

Capitalize the starting letter of every word in variable.

Eg: book="dont judge book by its cover"
print(book.title())

removeprefix():

Removes the given string/entrie string of the prefix in the variable

Eg: d="mukesh ambani"
print(d.removeprefix('mu')) #output: kesh ambani

removesuffix():

Removes the given string/entrie string of the suffix in the variable

```
Eg: d="mukesh ambani"
print(d.removeprefix('ambani'))      #output: mukesh
```

replace():

Replaces the old value with new value.

```
Eg: c="teacher he took my pen please ask him"
cl=c.replace('pen','pencil')
print(cl)
```

Below example have two 'is'. this and is. To change only 'is' to 'are' give exta space.

```
Eg2: c="this is my book"
cl=c.replace(' is',' are')
```

Above example 2 can be achieved using below:

```
Eg3: x="this is book"
c=x.split()
        for i in range(len(c)):
                if c[i]=='is':
                c[i]='are'
        print(c)
        d=' '.join(c)
print(d)
```

join():

join will convert the list to string while split converts the string to list.

```
Eg: x="this is book"
c=x.split()
        for i in range(len(c)):
        if c[i]=='is':
        c[i]='are'
                print(c)
        d=' '.join(c)
        print(d)
```

Task-13:

Practice total string topic

Task-14:

Prepare document on strings

Task-15:

Take a paragraph and replace ‘is’ with ‘are’.

Python - String & String Methods — Aug 08. Mon

Dictionary:

* dictionary is data type used to store data in key-value pair format.
* It is the mutable data-structure like list.
* keys must be a single element as string. int. tuple
* key must be unique. no duplicates allowed. Key is immutable.
* key. value separated with :(colon) and items separated with .(comma)
* values can be any type such as list. tuple. integers.dictionary.string. bool. complex. etc.
* key & value considered as a single item.
* dictionary is denoted with {} - empty dictionary

* dictionary can be defined in two ways:

Type1 : eswar={} # General use
print(type(eswar))

Type2: eswar=dict() # for type conversion
print(type(eswar))

Eg: ajay={'name':'kiran','rollno':20}
Here name=key. kiran=value

* We can't access dictionary items using index. Here key will act as index.

Eg: ajay={'name':'kiran','rollno':20}

* We can access list items using two-dimentional indexing either forward/backward index.

Eg: ajay={'name':'kiran','rollno':20.1:[0.1.2],'phone':[1234567]}
print(ajay[1][-1]) # print(ajay[1][2]) - forward index
Output: 2

Dictionary Methods:

clear copy get keys values items update pop

* clear and copy have same functionality as list functions

***get*():**

* Used to get the value for a key like above indexing exmple.
* Method have fast execution when compare with index usage.

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
print(ajay.get('roll'))

***keys*() & *values*():**

* keys() and values() will give the list of keys/values present in the variabe.

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
print(ajay.keys())
print(ajay.values())

***items*():**

* Return both keys and values in tuple format.

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
print(ajay.items())

***update*():**

* Can insert new key-value pair at the end of variable items. Also can update the existing value of a key.

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
ajay.update({'address':'hyd'})
ajay.update({'name':'james bond'})

***pop*():**

* Removes the given key pair from the variable

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
ajay.pop('name')
print(ajay)

* keys() returns only keys and values() returns only values. items() will return keys and values in tuple format.

Eg: ajay={'name':'kiran','rollno':20,l:[0,1,2],'phone':[1234567]}
for i in ajay:
print(i)

* To get the keys and values without tule braces:

```
ajay={'name':'kiran','rollno':20.1:[0.1.2],'phone':[1234567]}
for i,j in ajay.items():
print(i,j)
```

Nested dictionary:

* a dictionary inside another dictionary is nested dictionary.

* nested dictionary values can access using keys.

```
Eg: eswar={
        1:'a',
        2:'b',
        3:{'adi':'kiran'}
        }
        print(eswar[3]['adi'])
```

* In real time, we use dictionaries in many places. In supermartkets billing we use item & price pair.

****

Tuple:

* Used to store the sequence of immutable data.
* Its same as list except one. data in list can change but can't in tuple
* value of the tuple cannot be changed
* tuple can be defined using paranthesis() and each element separated by comma(,).
* Can use index in tuple. As indexing supports, must can use slicing.

tuple declaration:

* tuple declaration is of two types:

```
sai=()
print(type(sai))

sai=tuple()
print(type(sai))
```

```
Eg: sai=(2,3,4,5,'kiran')
        print(sai)
```

tuple index:

* Can get tuple values using index number:

Eg: sai=(2.3.4.5.'kiran')
print(sai[l])

tuple built-in functions:

* No functions for tuple except built-in ones (min. max. sum. len).

Eg: sai=(2.3.4.5)
print(max(sai))
print(min(sai))
print(sum(sai))
print(len(sai))

* these built-ins can be used for list. tuple. dictionaries also.

tuple operations:

* By using the built-in functions. can perform below operations.

l. repetition
2. concatination
3. membership
4. iteration

l. repition:

* Repeats printing the entire tuple

Eg: mounikal=(l.2.3.6.l.2.6.3.6.66)
print(mounikal*3)

2. concatination:

* Adds two tuples

Eg: tl=(l.2.3)
t2=(4.5.6)
print(tl+t2)

3. Membership:

* Checks the condition and results True/False using in. not in.

Eg: mounika=(1.2.3.6.1.2.6.3.6.66)
print(100 in mounika)

Eg: mounika=(1.2.3.6.1.2.6.3.6.66)
print(100 not in mounika)

4. Iteration:

* Iterates the values passed in tuple

Eg: mounikal=(1.2.3.6.1.2.6.3.6.66)
for k in mounikal:
print(k)

****

Task-16:

(i) Practice total dict and tuple (ii) prepare document on both types

Task-17:

start python interview questions

Task-18:

practice python programs in other websites

Websites to practice python programs:

* w3schools python exercises
* hackerrank python
* codechef python

##########

Mini Project: Bank Management System

1. Ministatement
2. Credit
3. Debit
4. user details

Step1 :

Take some users in dictionary.

Step2 :

Check login credentials

Step3 :

Take user input for either ministatement. crdit. debit or account details

Step4 :

Result

Python - Sets & Frozen Set — Aug 10. Thur

Sets:

* Elements in the set cannot be duplicate(if duplicates given. it will take but only prints all elements except duplcicates).
* Elements in the set are mutable.
* As no indexing for elements in set. no slicing support.
* Can add/remove items from set.
* Set can be defined with {}
  Set - values separated with comma(.) in {}.
  dictionary - key & value pairs separated with comma(.) in {}
* Can use built-in set() for declaration. (same like built-in functions for other data types)

```
set declaration:  d={}
                  print(type(d))
```

* By default. it will show the type as dictionary. Based of the values in {}. it will decide whether set or dictionary.

Set functions:

add clear copy pop remove update

add():

* Adds the given value at the end of the set. As there is no indexing in set. can't add new element in middle.
* It will work same as append().

```
Eg: d={1.2.3.4.4.2.4.2.4}
    d.add('kiran')
    print(d)

    output: {1.2.3.4.4.2.4.2.4.'kiran'}
```

pop():

* will deletes smallest element in the set. But not remove negative numbers and strings if there is any.

```
Eg: d={1.2.3.4.0.-1.4.2.4.2.4.'kiran'}
    d.pop()
    print(d)

    output: {1.2.3.4.-1.4.2.4.2.4.'kiran'}
```

remove():

* removes given element from set

Eg: d={1.2.3.4.0.-1.4.2.4.2.4.'kiran'}
d.remove(-1)
print(d)

output: {1.2.3.4.0.4.2.4.2.4.'kiran'}

update():

* adds given elements to the set randomly. Because set is unordered.
* update will add multiple elements but add() cannot support for this.

Eg: d={1.2.3.4.0.-1.4.2.4.2.4.'kiran'}
d.update({22.33.44})
print(d)

output: {1.2.3.4.22.0.-1.4.2.33.4.2.44.4.'kiran'}

Set Operations:

* We can perform below operations using sets. We need minumum 2 sets to perform them.

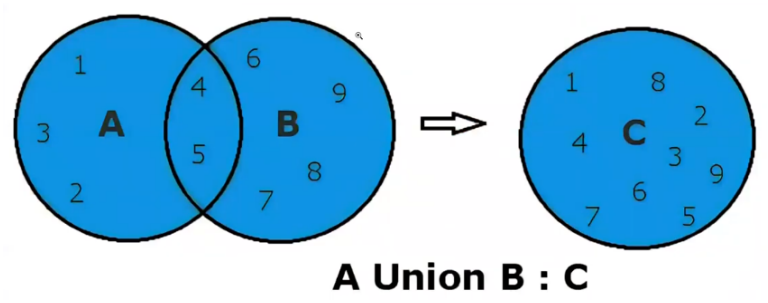
1. union
2. intersection
3. difference
4. symmetric difference
5. issubset
6. isdisjoin
7. issuperset

union():

* Combinely represents the elements of both sets by removing duplcicates

Eg: sl={1.2.3.4.5}
s2={3.4.5.6.7}
print(sl.union(s2))

output: {1.2.3.4.5.6.7}

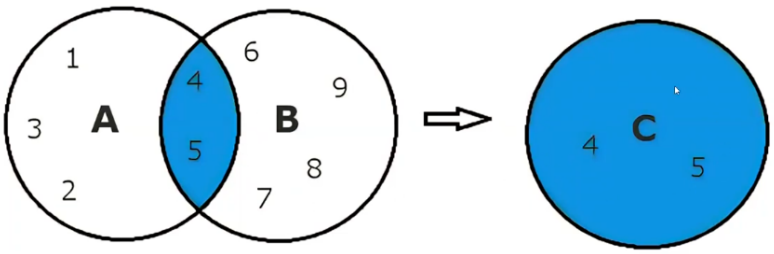


A Union B : C

***intersection()*:**

* Display common elements in both sets.

Eg: sl={1.2.3.4.5}
s2={4.5.6}
print(sl.intersection(s2))



A Intersection B : C

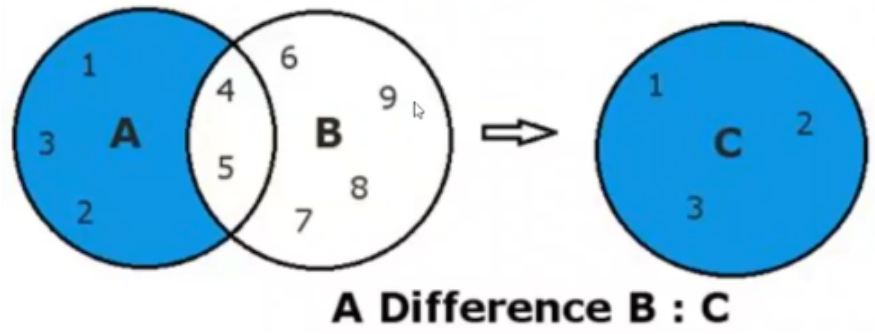
output: {4.5}

***difference()*:**

* Substracts common elements and second set elements. Result will be the remaining first set elements.

Eg: sl={1.2.3.4.5}
s2={4.5.6}
print(sl.difference(s2))

output:{1.2.3}



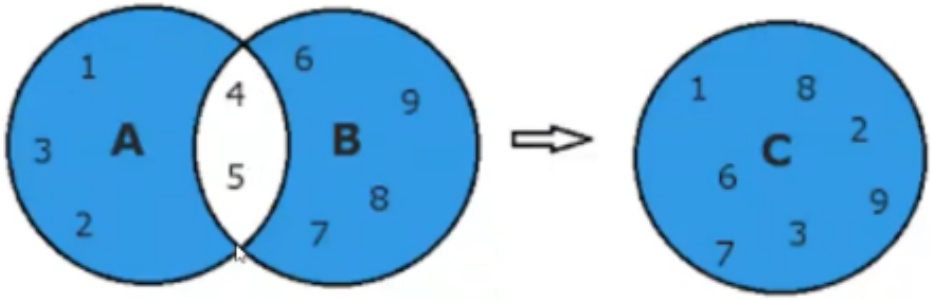
A Difference B : C

***symmetric_difference()*:**

* Returns all the elements except common ones in both sets.
* Exact opposite to intersection.

Eg: sl={1.2.3.4.5}
s2={4.5.6}
print(sl.symmetric_difference(s2))

output: {1.2.3.6}



A Symmetric Difference B : C

***issubset* & *issuperset*()*:**

* If all the elements of sl present in s2(can have others also). Then sl called as subset of s2 and s2 called superset of sl.
* both returns True/False

Eg: sl={1.2.3.4.5}
s2={1.2.3.4.5.6}
print(sl.issubset(s2))

output: True

Eg: sl={1,2,3,4,5}
s2={1,2,3,4,5,6}
print(s2.issubset(sl))

output: True

isdisjoint():

* Both sets have no common elements, then result will be True else False.

Eg: sl={1,2,3,4,5,6}
s2={7,8,9}
print(sl.isdisjoint(s2))

otput: True

frozen set():

* Can convert mutable type to immutable.
* once below list converts into frozen set, we can't perform any operations except built-in functions.

Eg: a=[1,2,3,4,5] # **mutable**
a.append(3)
print(a)

Eg2: a=[1,2,3,4,5] # **immutable**
c=frozenset(a)
print(c)

list items addition:

zip():

* Can add two list items using zip().zip will generate an object, convert it to get tuple (combination of both list items).

Eg: a=[1,2,3,4,5]
b=[2,3,4,5,6]
d=zip(a,b)
print(d)
print(list(d))
c=[i+j for i,j in zip(a,b)]
print(c)

output: [3,5,7,9,11]

****

Task-19:

Practive set and frozen set

Task-20:

Prepare a doc on total datatypes

Task-21:

Explore random module and date time module

Python - Functions & Advanced Functions — Aug 11. Thu

Functions:

* Functions is a block of code which runs only when it is called. We can write code once and reuse it whenever needed.

Function definition:

* Function contains definiton. body. other functions. variable. loops etc. We can define and call function as follows:

definition syntax: def function_name():
calling syntax: function_name()

```
def function_name():                    # function definition
        # function body
        # function
        # variable
        # for loop
```

```
Eg: def satish():                       # function definition
        print('this is james bond')
    satish()                            # function call
```

Parameters & Arguments:

* Values passing to function definition are called parameters. For a single parameter. we can pass either single or multiple arguments(through arbitary argument & keyword argument).

* values passing to parameters through function call are arguments.

* for loop usage in functions:

```
Eg: def satish():
        print('this is james bond',a,b,c)
    for i in range(100):
    satish(100,1,2)
```

* Multiple arguments passing to single parameter using arbitary argument. '*' is used before the parameter to do this and output will be tuple.

```
Eg: def ajay(*a):                       # arbitary argument passed(*a)
        print('this is ajay',a)
   ajay(100,1,23,4,5,6,7,8,9,0)
```

output: (100.1.23.4.5.6.7.8.9.0)

* We can pass arguments to parameter through keyword argument to get output in dictionary format. '**' is used before parameter to achieve this.

```
Eg: def ajay(**a):                                  # keyword argument passed(**a)
        print('this is james bond'.a)
    ajay(name='kiran'.age=50)
```

output: {'name':'kiran'.'age':50}

Docstring(3rd type of comments):

* We use docstrings only inside functions as below:

```
""" This is Doc String """
```

* We can't print other type of comments (# and ''' ''') but can print docstring using doc construction (__doc__) (two underscores both sides of doc)

```
Eg: def simha():
        """ This is Doc String """
        print('this is fun')
    print(simha.__doc__)
```

User-defined modules:

* We can create some functions in a file and can import them to other python files while programming. This is done through user-defined modules.

* functions written in a python file (functions.py) imported in other file (user_module.py).

import syntax: from functions import *

* If need to import all functions from user-defined module(functions.py.). give * otherwise mention required function name alone.

Eg: we can import mult() instead of *

return vs print:

* Generally programmers use return/print statements at the end of a function.

* return will not generate output after calling function. function call need to assign a variable and need to print that variable for output.
* function will terminate/killed after return usage statement.
* advantage of return is to avoid printing unwanted statements in output.

Eg: add(22.22) # print usage in module.

* print directly generates output without help of separate variable.

```
Eg: v=add(22.22)
    print(v)                    # return usage in module.
```

lambda function:

* lambda is a small anonymous function. It can take any number of arguments with single expression. As this is a function. its output will display only when we call it.
* As lambda is a nameless function. we assign it to a file object(x here) and treat this as a function name. lambda used to write function in single line.

```
Eg: x=lambda a.b.c: a+b+c
    print(x(5.2.22))
```

* x is file object. lambda is a nameless/anonymous function.

```
Eg2: a=[12.34.5.6.8]
     b=[ ]
     for i in a:
          t=lambda c:c+1
          b.append(t(i))
     print(b)
```

Filter():

* Filter method filters the given sequence with the help of function that tests each element in the sequence whether true or not.

Eg:
```
ages=[5,12,17,18,24,32]
def myFunc(x):
    if x>=18:
        return True
    else:
        return False

adults=filter(myFunc,ages)
print(list(adults))
```

* filter() only filters the given data and data will not undergone into any changes. filter() will generate an object like zip() method and we need to convert it using list.

map():

* map() function changes each item of an iterable (list, tuple etc.) using function logic if given condition is True.

syntax: map(function, iterables)

Eg:
```
def calculateAddition(n):
    return n**2

numbers=(1,2,3,4)
result=map(calculateAddition,numbers)
print(list(result))
```

output: [1,4,9,16]

generator():

* generator defined as normal function used to generate output whenever required. It have 'yield' keyword rather than return.
* If body of a function definition contains yield, then its automatically becomes a generator function.
* function will terminate if 'return' used, but 'yield' will just holds execution without termination.

Eg:
```
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3
x=simpleGeneratorFun()
print(x.__next__())
```

* yield statement responsible for controlling the flow of generator function. It pauses the function execution by saving all states and yielded to the caller. It resumes the execution when a successive function is called.
* We can use only one return statement in a function but can use multiple yield statement in a generator function.

reduce():

* reduce() will returns a single value by compressing function and variables. We can't use reduce() like other functions. It needs to import from 'functools'.

* The first two elements of the sequence are choosen in the first step. and the result is archieved. The saved result will assign next time and after applying the same function to the previously obtained result and the number preceding the second element. This method is repeated until end of the elements container. Final result returned to the console and will print.

Eg:
```
from functools import reduce
d=reduce(lambda a,b: a+b,[23,21,45,98,2])
print(d)
```

* first 23 will store in a. 21 in b. a+b=44. Then 44 store in a. 45 store in b. a+b=89. Then 89 store in a and 98 in b and so on....

****

Task-22:

Practice function and advanced functions

Task-23:

Prepare doc on advanced functions

Task-24:

Explore about command prompt.

Python - File & Error Handling — Aug 12. Fri

File Handling:

* File handling is nothing but handling open. read. write and other operations performing on the file.

flow: open --> read/write --> close

File Modes:

Three types of file modes.

r	-read mode
w	-write mode
a	-append mode
r+	- at a time we can read and write
w+	- at a time we can write and read

read mode:

* We can just read the data in the file. Will not allow to do modifications.

Eg:
```
f=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='r')
c=f.read()
print(c)
f.close()
```

write mode:

* previous data in the file is replaced with given data if we use write mode. As replacement takes place. there will be some data loss.

Eg:
```
f=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='w')
c=f.write('this is write function')
f.close()
```

append mode:

* append is also a type of write mode but without data loss. It adds the given data at the end of original one.
* we just use mode='a' but the function will be write() only.

Eg:
```
f=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='a')
c=f.write('this is append function')
f.close()
```

readline/readlines:

* we can read all the data in a file using read() but can't see as per our requirement and can't insert some data into it. This can be achieved using readline/readlines.

* if empty readline() given. it will show first line of whole data in file.
* if readline(20) given with some number. it will show the data upto that number of charactrers.
* if readlines() used. all the data in file converts into list. So that we can easily inset and modify data into it as per our requirement.

Eg:
```
f=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='a')
c=f.readline()
f.close()
```

tell() function:

* tell() method used to get the position of file handle. It returns current position number of handle. It takes no parameters and returns an integer.
* File handle is like a cursor. which defines from the data has to be read / write in the file

Eg:
```
fp=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='r')
print(fp.tell())
fp.close()
```

Eg2:
```
fp=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='r')
fp.read(5)
print(fp.tell())
fp.close()
```

seek() function:

* seek() used to change the position of file handle to given position.

.

Eg:
```
fp=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='r')
fp.read(8)
print(fp.tell())
fp.seek(0)
print(fp.tell())
fp.close
```

Eg2:
```
file=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='r+')
content=file.read()
v=str(content)
f=v.split()
print(f)
f.insert(2,'chetan')
print(file.tell())
file.close()

file=open('D:\DS Practice\Aug12 - Error handling\demo.txt',mode='w')
for i in f:
file.writelines([i])
```

Research:

binary mode. w+. a+

Error Handling:

* Error handling can be done through the below blocks. If this technique used. only error message will notify you but not impacts execution of remaining code.

  try except else finally

* except is mandatory while try used. Can use as many 'except's in single try.

* we write error probability/risky code in 'try' block. message that need to return if error occured will be in 'except' block. message if no error came. will be in 'else' block. 'finally' block will execute irrespective of try and except cases.

syntax:
```
try:
        print('a')
except:
    print('error vachindi')
else:
    print('error raledu')
finally:
    print('nenu epudina vastanu')
```

Eg:
```
try:
        print(l+'kiran')
except Exception as r:      # Exception used. will print error msg.
        print('r')
```

Eg2:
```
try:
        print(a)                # use multiple try blocks if needed.
        print(b)
except NameError:
        print('a')
except ValueError:
        print('b')
```

raise(user defined error):

* It is used to create a user-define error.

Eg:
```
s='kiran'
try:
        number=int(s)                   # risky code
except ValueError:
        raise ValueError('string cannot convert into int')
```

Types of Errors:

indentation error	
eof error	
value error	
key error	- index trail on dictionary.
name error	- if variable not defined.
file not found error	- if file not there in given path.
module not found error	- if module not there.
index error	- if 10 elements there but calling 11th index.
keyboard interupt	- if forcefully stop execution using keyboard.
os error	- while using os module.
type error	- if string given in place of int
zero division error	- something / = infinity
syntax error	- if taken syntax wrong.
import error	- same like module not found error.

****

Task-25:

practice file handling and error handling

Task-26:

Explore os module (mkdir. cd. rm. list)

Task-27:

Prepare interview questions upto present syllabus.

Python - OOPs Basics

Aug 15. Mon

POP (Procedure Oriented Programming):
- Function based programming approach.

OOP (Object Oriented Programming):
- Object based programming approach.

OOP vs POP:

* Earlier days people used POP approach but nowadays migrated to OOP .

OOP	POP
programs are divided into parts known as objects	1.In this programs are divided into functions
2. The main focus of oop is on the data	2.The main focus of POP is on the procedures
3.It follows bottom-up approach	3. It follows top-down approach
4.It has access specifiers such as public,private etc.	4.In this most functions use global data
5.It provides data hiding,data associated with the program.so security is provide	5.POP does not have any access specifiers
6.Ease of Modification	6.POP does not provide any data security
7.Examples:Java,Perl	7.Modification is difficult.
	8.Examples: C,COBOL etc..

OOPs Features:

1. Class
2. Object
3. Inheritance
4. Polymorphism
5. Data Abstraction
6. Encapsulation

Class:

* Class is a collection of objects. It's like blue print/template to create an object. We can define a class by using 'class' keyword. First letter of class name should be capital letter. class body contains constructors. functions.

variables.

Eg: fruits # class
Eg: apple. orange. mango # objects

defining class: class Pramod(): # () is optional

Object:

* Object is a physical/real entity. It's an instance of class. Memory is created to class only when objects declared. We can create multiple objects for a single class and class variables can be accessed by different objects.

class access: objectname=classname()
method access: objectname.method()

* attributes/data members in OOP are same as variables in POP approach.

* methods/member functions in OOP are same as functions POP.

self:

* We can access attributes and methods of the current class using 'self' keyword. self keyword should be taken as first parameter by default in method.

```
class james():                          #james() is class
        def display(self):              # display() is method
                print('this is class')
obj=james()                             # obj is object
obj.display()
```

* variable of a function can be directly access but can't be accessable from other function. Thi can be achieved through 'self' keyword.

constructors:

* class name and constructor name should not be same.
* python will not support multiple constructors.
* constructor also treated as method in python(can use def keyword)
* constructor will execute immediate after object creation. without calling.

__init__ :

* init refers initialization. 'init' used to initialize all datamembers at a place and use them throughout the class.
* As variables declare and initialize at program starting in C, C++, python use __init__ for the same purpose.
* variables declared globally can accessable throughout class but all variabels should not declare as global variables.

****

Task-28 :

Practice class,object,init,self

Task-29 :

Prepare a doc on basics of oops

Task-30 :

Finish interview questions on basic oops

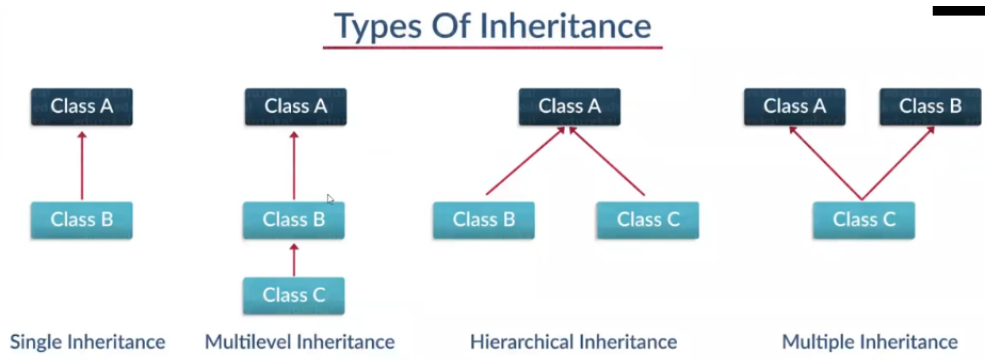
Python - OOPs advanced concepts — Aug 16. Tue

- adibogadi

Inheritance:

The process of accessing properties from parent/ Base/ Super class to child/ derived/ sub class.

Types of inheritance:



1. Single Inheritance –
   a derived class acquires the members of a single super class.
2. Multi-level inheritance –
   a derived class dl inherited from base class basel and d2 inherited from dl.
3. Hierarchical inheritance –
   from one base class we can inherit any number of child classes
4. Multiple inheritance –
   a derived class is inherited from more than one base class

Hierarchical type is quite opposite to multiple inheritance. we can take __init__ in single class if needed.

Polymorphism:

Polymorphism is a Greek word. Poly means 'many' and morph means 'forms'.

Types of Polymorphism:

1. Methodoverloading
2. Methodoverriding

Method overloading:

'None' is a default parameter in Method overloading program. As we didn't passed all parameters to the function. it will throw error. If we use 'None'. it will returns default 'None' instead of error.

Eg:
```
class Methodoverload:
        def something(self.a=None. b=None. c=None):
                print(a.b.c)
obj=Methodoverload()
obj.something(1.2.3)
obj.something(1.2)
obj.something(1)
obj.something()
```

There are different approaches to overcome the method overloading. using default parameters is one of them.

Method overriding:

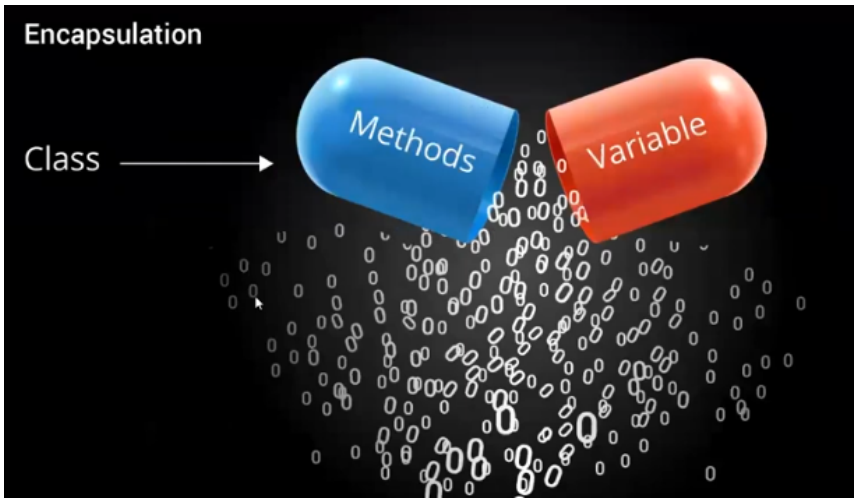
If both child & parent have same function name. when we call function using object. it will reflect lastest child function only. It will override parent function.

Eg:
```
class Methodoverrid:
        def display(self):
                print('this is parent class')
class Methodover(Methodoverrid):
        def display(self):
                print('this is father class')
                super().display()
class Child(Methodover):
        def display(self):
                print('this is child class')
                super().display()
obj=child()
obj.display()
```

we can use 'super()' to access the parent function in this scenario. while 'self' used to access same class methods and variables. 'super()' used to access parent methods and variables.

Encapsulation:

Binding of class using methods and variables(attributes) is called Encapsulation.



access specifiers/modifiers:

access modifiers will provide security.

public - anyone can access
and
private - only one can access (__)
protected - inherited access (_) (grandfater car can use that family)

Data Abstraction:

It's nothing but hiding of data. But nowadays people not hiding in this type. 'abs' method have no body. we can use 'pass'. need not to create object for abs class. a class contain one or more abstract methods. then it is said to be abstract base class (abc).

****

Task-31 :

Practice total oops and solve exercises in different websites

Task-32 :

Prepare a document on OOPs.

Task-33 :

Research OOPs based projects.

Python - Web Scrapping

Aug 17. Wed

Web Scrapping:

Web Scrapping is the process of extracting data into structured format from webpages. It helps to collect unstructured data and later we can convert it into a structured manner.

pip:

pip is the standard package manager for python. It allows you to install and manage additional python packages. we can use alias for the importing packages using 'as'

```
pip install pandas as pd
pip install requests
pip install beautifulsoup4
```

Pandas:

Pandas used to store the scrapped data into a table(webframe) neatly.

Requests:

Request module used to send requests to websites to scrap data.

beautifulsoup4:

Methods of beautifulsoup4 is used in web scrapping.

Webscrapping Implementation:

1. **Requesting website:**

```
response=requests.get("webpage url")
print(response)
```

if the output for response=200 means website accepting our request. So that we can proceed for scraping. Else can't proceed further on that website.

2. **Collect raw data:**

```
soup=BeautifulSoup(response.content,'html.parser')
```

BeautifulSoup - to extract data.
html.parser - parsers will convert the requested data.

As website in html. we used html parser. If it is in json. can use json parser.

3. Classes Extraction:

```
names=soup.find_all('div',class_="_4rR01T")
```

names - variable name to hold collected mobile names data.
soup - already have scrapped raw data.
find_all - differentiate required class among others.
div - html attribute which holds the class.

right click on any mobile name and choose 'inspect' in webpage. take that class name and html tag in the above format.

4. Collecting particular data:

(i) Mobile Names Scrapping:

```
names=soup.find_all('div',class_="_4rR01T")
name=[]
for i in names:
    d=i.get_text()
name.append(d)
```

name=[] - empty list to collect modified/transformed data in later steps.
get_text() - extracts only text by ignoring html tags and others.
append() - adds the transformed data to the empty list. already taken.

(ii) Prices Scrapping:

```
prices=soup.find_all('div',class_="_30jeq3 _1_WHN1")
price=[]
for i in prices:
    d=i.get_text()
    s=d[1:]
    price.append(s)
```

[1:] - slicing used to ignore 0 index character(rupee symbol).

Price data come with comma. We can remove this using later classes.

(iii) Ratings Scrapping:

```
rates=soup.find_all('div',class_="_3LWZlK")
rate=[]
for i in rates:
    d=float(i.get_text())
    rate.append(d)
```

(iv) Images Scrapping:

```
images=soup.find_all('img',class_="_396cs4 _3exPp9")
image=[]
for i in images:
    d=i['src']
    image.append(d)
```

img - html element that holds images.
src - img tags wll have src(source) attribute.

As images are not text, we can use ['src'] to get image source links.

(iv) Links Scrapping:

```
links=soup.find_all('a',class_="_1fQZEK")
link=[]
for i in links:
    d="https://www.flipkart.com"+i['href']
    link.append(d)
```

a - anchor tag that holds links in html page.
+ - concatinates missed url with scrapped one.

Links not generated fully. Its depends on the website. In this scenario, we can concatinate missed url.

5. Take scrapped data into table(DataFrames):

Now we have all data and need to save it into a table(DataFrame). Now, we have to use pandas.

```
df=pd.DataFrame()
df['Name of Mobile']=name
df['Mobile Price']=price
df['Mobile Ratings']=rate
df['Mobile Images']=image
df['Mobile Links']=link
df.to_csv("Mobile_data.csv")
```

DataFrame - to store data in two-dimentional way(columns and rows).
df.to_csv - to export data in data frame to csv file.

```
df['Name of Mobile']=name[10]
```

If any error thrown due to structure of data, use [10] (some number) at the end and excute it. Next just remove [10] and execute.

We didn't done automation here. So if any data changed in flipkar site, will not take change until we execute the program again.

WS Realtime scenarios:

* Hotel trivago have no hotels but they scraps other websites for data.
* Single employee can run news website by just scrapping others sites.

Machine Learning

To teach/train anything to a machine is called machine learning.

Types:

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

1. Supervised learning:

We have to give both input and output for this type.

2. Unsupervised learning:

We jus give input, system need to generate output based on model.

3. Reinforcement learning:

Tesla car is example. Once it is steps into field, it will learn and react as per the situation and takes decision instantly. This type of learning is still under research till not came into existance .

Machine learning lifecycle:

step-1: (kaggle)

data gathering

step-2: (panda, numpy, stats or can use ETL tools)

data preprocessing/transformation (unstructured ----> structured)

(i) data cleaning
(ii) data transformation
(iii) data reduction.

step-3: (matplotlib. seaborn. powerBI/Tablueau)

data visualization (data ----> graphical)

step-4: (sklearn)

(i) Feature extraction - decide to use which column on which type.
(ii) Model selection - can take choice of model and to train that one.
(iii) Training - Out of 100% data. 70% used for training and 30% for testing.
(iv) Testing - Its good model if resulted accuracy is 80-90% . Extreme high and lows are dumb/waste models.
(v) Prediction - Based on the result. we can predict the future trend.

step-5: (django. flast. sql. aws or azure or gcp. github)

deployment

* Worked model need to release in the market. We can deploy using aws. git or any other to publish it out.

****

Task-34:

Extract data from 3 pages using webscrapping

Task-35:

Prepare a documentation on machine learning steps

Task-36:

Install anaconda navigator and explore jupyter notebook

In [1]:

```
!pip install numpy
import numpy as np
```

```
Requirement already satisfied: numpy in c:\users\administrator\anaconda3\lib
\site-packages (1.21.5)
```

Numpy

Numpy means Numerical Python. It is a python library. It consists of multi dimensional arrays.

Array vs List:

Arry is fast than List.

Python List:

1. Able to store different data types in single list.
2. Storing each item in random location in the memory.
3. Good for the scenario where list can grow dynamically.
4. Inbuilt data type (need not import any library)
5. It has more inbuilt functions.

NumPy Array:

1. Can store only one data type in an array at any time.
2. Storing each item is sequential which makes array more effective in processing.
3. Good for the scenario where the items are fixed size and same data time.
4. Need to install external library NumPy.
5. No extra functions, so it will not more momory store.

Array Dimensions

An array contains layers,rows and columns. 4 columns in 1D array diagram are considered as single row.

axis 0 represent rows and axis 1 for columns.

shape: (4,3,2,) represents 4 layers, 3 rows and 2 columns.

we can install numpy using - !pip install numpy/pip install numpy

we will import numpy same as pandas like - import numpy as np

Defining Array

In [7]:

```
# defining array

import numpy as np
a=np.array([1,2,3,4,'a'])
a                                # prints a
```

Out[7]:

```
array(['1', '2', '3', '4', 'a'], dtype='<U11')
```

```
Array will take all elements of same data type. But here we gave int and string.

So once we run, it will converts all data into string. Because string can't changed to int
but int can change to string.
```

ndim

In [13]:

```
# ndim used get the dimention of an array.

a=np.array(10)
b=np.array([1,2,3])  # one dimention
c=np.array([[1,2,3],[4,5,6]])   # two dimention
d=np.array([[[1,2,3],[4,5,6]], [[1,2,3],[4,5,6]]])  # three dimention
print("-----------")
print(a)
print(a.ndim)
print("-----------")
print(b)
print(b.ndim)
print("-----------")
print(c)
print(c.ndim)
print("-----------")
print(d)
print(d.ndim)
```

```
-----------
10
0
-----------
[1 2 3]
1
-----------
[[1 2 3]
 [4 5 6]]
2
-----------
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
3
```

shape

In [14]:

```
# shape used to check shape (layers,rows and numbers). layers if any divided with single li

# [ - no of initial braces tell the dimention of array.     ('[[['

print("-----------")
print(a)
print(a.shape)
print("-----------")
print(b)
print(b.shape)
print("-----------")
print(c)
print(c.shape)
print("-----------")
print(d)
print(d.shape)
```

```
-----------
10
()
-----------
[1 2 3]
(3,)
-----------
[[1 2 3]
 [4 5 6]]
(2, 3)
-----------
[[[1 2 3]
  [4 5 6]]

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

dtype

In [15]:

```
# dtype - used to get data type. it will show like int32. just ignore of number 32. What we

print(a.dtype)
print(b.dtype)
print(c.dtype)
print(d.dtype)
```

```
int32
int32
int32
int32
```

type()

In [16]:

```
# type - will tell the type of the class

print(type(a))
print(type(b))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

arange()

In [20]:

```
# arange - this is used as range function in other data types

print(np.arange(1,11))
print(np.arange(1,11,2))
```

```
[ 1  2  3  4  5  6  7  8  9 10]
[1 3 5 7 9]
```

eye()

In [21]:

```
# eye - used to get Identity matrix with mentioned dimention. default data type is float.

# eye() usage - Identity matrix
np.eye(3,dtype=int)
```

Out[21]:

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

zeros()

In [22]:

```
# zeros - gives all 0 matrix with given dimention

# zeros() usage - null matrix
np.zeros((4,3),dtype=int)
```

Out[22]:

```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

ones()

In [23]:

```
# ones - gives all 1 matrix with given dimention

# ones() usage - unitary matrix
np.ones((3,5))
```

Out[23]:

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

full()

In [24]:

```
# full - gives matrix with given number and with given dimention

# full() usage
np.full((4,3),10,dtype=int)
```

Out[24]:

```
array([[10, 10, 10],
       [10, 10, 10],
       [10, 10, 10],
       [10, 10, 10]])
```

diag()

In [25]:

```
# diag - prints given numbers in diagnal while other elements 0

# diag() usage
x=[1,2,3,4,5,6,7]
np.diag(x)
```

Out[25]:

```
array([[1, 0, 0, 0, 0, 0, 0],
       [0, 2, 0, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0, 0],
       [0, 0, 0, 4, 0, 0, 0],
       [0, 0, 0, 0, 5, 0, 0],
       [0, 0, 0, 0, 0, 6, 0],
       [0, 0, 0, 0, 0, 0, 7]])
```

Reshape:

In [27]:

```
# shape will gives the elements of current shape. We can change it using reshape.

# suppose original matrix is 3x4. so we have 12 elements. we reshape them as 6x2(=12),2x6,4

x=np.arange(1,17)
a=x.reshape((4,2,2))
print(a)

# This will gives a matrix with 4 layers, 2 rows and 2 columns. Totally 1-16 digits.
```

```
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
  [11 12]]

 [[13 14]
  [15 16]]]
```

ravel()

In [28]:

```
# ravel - this is same as undo. this is used to undo the reshaped matrix into previous dime

a=np.arange(1,11)
s=a.reshape((5,2))
z=s.ravel()
print(z)

# z=np.ravel(s) - we can use above line or this line for using ravel.
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

flatten()

In [29]:

```
# flatten - if data in any dimention, flatten will change it to 1 dimention.

a=np.arange(1,17)
print(a)
b=a.reshape(8,2)
print(b)
c=b.flatten()
print(c)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
[[ 1  2]
 [ 3  4]
 [ 5  6]
 [ 7  8]
 [ 9 10]
 [11 12]
 [13 14]
 [15 16]]
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
```

slicing

In [32]:

```
# slicing - We can use slicing same like in other data types.

# We can access particular element of matrix like below:

a=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])        # (2x2x3)array creation
a
# We can access element 8 as: a[1][0][1]
```

Out[32]:

```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

numpy copy vs view:

```
copy is new array and view is just to see original array
        a=10, b=a;  Here b is a's view
copy owns data and any changes done to copy will not affect original.
```

shares_memory

In [36]:

```
x1=np.arange(10)
print(x1)
x2=x1                    # x2 is x1 view
print(x2)

# This can be checked using. so it will give result True.

print(np.shares_memory(x1,x2))

# As x1 and x2 data is same, if we check their id's, it will give same memory. this phenome
```

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

Transpose of array

In [37]:

```
# Rows will change to columns and columns to rows.

# transpose()
y=np.array([[1,2,3],[4,5,6]])
print(y)
print(y.transpose())
```

```
[[1 2 3]
 [4 5 6]]
[[1 4]
 [2 5]
 [3 6]]
```

hstack vs vstack function

In [38]:

```
# hstack() - If two arrays taken into hstack, resulted array will be flattern array

# vstack() - If two arrays taken into vstack, resulted array will be in vertical

# hstack and vstack
p=[1,2,3]
q=[4,5,6]
r=[7,8,9]
print(np.hstack((p,q,r)))
print(np.vstack((p,q,r)))
```

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

insert and delete

In [42]:

```
# we can add element at a given index in an array using insert.

print(x1)
print(x2)
print(np.insert(x1,4,x2))
print(np.delete(x1,1))
# we can delete a particular element at given index using delete.
```

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

trigonometry

In [44]:

```
# We can perform trigonometric and exponential operations such as sin,cos,tan,cot for any g

# trigonometric operations
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print("sin values: \n",np.sin(a))
print("cos values: \n",np.cos(a))
print("tan values: \n",np.tan(a))
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
sin values:
 [[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155 ]
 [ 0.6569866   0.98935825  0.41211849]]
cos values:
 [[ 0.54030231 -0.41614684 -0.9899925 ]
 [-0.65364362  0.28366219  0.96017029]
 [ 0.75390225 -0.14550003 -0.91113026]]
tan values:
 [[ 1.55740772 -2.18503986 -0.14254654]
 [ 1.15782128 -3.38051501 -0.29100619]
 [ 0.87144798 -6.79971146 -0.45231566]]
```

Sum

In [50]:

```
We can sum the elements in array. Can add row wise and column wise also

# various sum types
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.sum(a))
print(np.sum(a,axis=0))
print(np.sum(a,axis=1))
```

```
  Input In [50]
    We can sum the elements in array. Can add row wise and column wise also
       ^
SyntaxError: invalid syntax
```

Statistical Operations

In [53]:

```
# We can perform statistical operations(median, mean, st.deviation) also.
# statistical operations
print(np.median(a))
print(np.mean(a))
print(np.std(a))
print(np.max(a))
print(np.sort(x2))
```

```
5.0
5.0
2.581988897471611
9
[0 1 2 3 4 5 6 7 8 9]
```

where

In [56]:

```
# where can be used to give condition in like sequential
m=np.where(x1 == 4)
print(m)
c=np.array([44,55,66,88,99,12,12,12,12,12,10,10,10,101,0])
print(np.where(c%2 == 0))

# replacing with 0 for the given condition
print(np.where(x1>5,x1,0))
```

```
(array([4], dtype=int64),)
(array([ 0,  2,  3,  5,  6,  7,  8,  9, 10, 11, 12, 14], dtype=int64),)
[0 0 0 0 0 0 6 7 8 9]
```

In [49]:

```
!pip install pandas
import pandas as pd
import numpy as np
```

```
Requirement already satisfied: pandas in c:\users\administrator\anaconda3\li
b\site-packages (1.4.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\administrator\anaco
nda3\lib\site-packages (from pandas) (1.21.5)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\administra
tor\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\administrator\anacon
da3\lib\site-packages (from pandas) (2021.3)
Requirement already satisfied: six>=1.5 in c:\users\administrator\anaconda3
\lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

Pandas

In [ ]:

```
* Pandas is a Python library used for working with data sets.
* It has functions for analyzing, cleaning, exploring and manipulating data.

* Pandas is powerful for data analysis
* Pandas stands for "Python Data Analysis Library"
* Pandas is one of the most preferred and widely using tools in Data munging/wrangling

* wrangling - also called data cleaning, data remediation.
* data mungling - refers to a variety of processes designed to transform raw data into more
```

Pandas Use

In [ ]:

```
* Pandas allows us to analyze big data and make conclusions based on statistical theories.
* Pandas can clean messy data sets and make them readable and relevant.
* Relavant data is very important in data science.
```

pandas installation

In [ ]:

```
!pip instal pandas
import pandas as pd
import numpy as np

* first need to import required libraries then start to write code.
```

Series

In [ ]:

```
* A pandas series is like a column in a table.
* It is a one-dimensinal array holding data type of any type.
* Series default index is 0,1,2,3.....
```

In [2]:

```
# Series with default index
s1=pd.Series([23,24,25,12,32])
s1
```

Out[2]:

```
0    23
1    24
2    25
3    12
4    32
dtype: int64
```

In [3]:

```
# We can change default index as below
# Series with manual index
s2=pd.Series([23,45,67,12,34],index=['a','b','c','d','e'])
s2
```

Out[3]:

```
a    23
b    45
c    67
d    12
e    34
dtype: int64
```

In [4]:

```
# We can change data type of values in Series
# Series with float
s3=pd.Series([23,45,67,12,34],index=['a','b','c','d','e'],dtype='float')
s3
```

Out[4]:

```
a    23.0
b    45.0
c    67.0
d    12.0
e    34.0
dtype: float64
```

In [5]:

```
# We can create Series using dictionary. Here keys will act as index and values work as col
# creation of Series using dictionary
s4=pd.Series({'e':65,'f':43,'c':45})
s4
```

Out[5]:

```
e    65
f    43
c    45
dtype: int64
```

DataFrame

In [ ]:

```
* Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a tabl
```

In [6]:

```
# DataFrame Creation
d1=pd.DataFrame([43,54,65,76])
d1
```

Out[6]:

	0
0	43
1	54
2	65
3	76

In [7]:

```
d2=pd.DataFrame([[2,3,4],[4,5,6],[1,2,3]])
d2
```

Out[7]:

	0	1	2
0	2	3	4
1	4	5	6
2	1	2	3

In [8]:

```
# converting series into dataframe with defaults
d2=pd.DataFrame(s2)
d2
```

Out[8]:

	0
a	23
b	45
c	67
d	12
e	34

In [6]:

```
# dataframe with user defined column names
d3=pd.DataFrame([[2,3,4],[4,5,6],[1,2,3]],columns=['a','b','c'])
d3
```

Out[6]:

	a	b	c
0	2	3	4
1	4	5	6
2	1	2	3

In [11]:

```
# dataframe with user defined column names and index
d3=pd.DataFrame([[2,3,4],[4,5,6],[1,2,3]],columns=['a','b','c'],index=['x','y','z'])
d3
```

Out[11]:

	a	b	c
x	2	3	4
y	4	5	6
z	1	2	3

In [ ]:

```
* dictionary keys will change to index in Series. But keys will change to columns in DataFr
```

In [12]:

```
# creating DataFrame from list of dictionaries
dic=[{'alex':1,'joe':2},{'ema':5,'dora':10,'alice':20},{'ema':5,'dora':10,'alice':20}]
pd.DataFrame(dic,index=['a','b','c'])
```

Out[12]:

	alex	joe	ema	dora	alice
a	1.0	2.0	NaN	NaN	NaN
b	NaN	NaN	5.0	10.0	20.0
c	NaN	NaN	5.0	10.0	20.0

In [ ]:

```
* We will get 'NaN' where data is not present. NaN means null values.
* If null values in dataset, we should not train it to model, as it will treated as dump mo
```

DataFrame Operations

In [ ]:

```
* If we do any operation by assigning it to a variable it will be permanent or else its sco
```

In [13]:

```
d3
```

Out[13]:

	a	b	c
x	2	3	4
y	4	5	6
z	1	2	3

In [14]:

```
# accessing DF columns using index
print(d3['a'])
print(d3['b'])
```

```
x    2
y    4
z    1
Name: a, dtype: int64
x    3
y    5
z    2
Name: b, dtype: int64
```

In [7]:

```
# adding new columns with mathematical operations
# d3 is DataFrame here
d3['d']=d3['a']*d3['b']
d3['e']=d3['a']+d3['d']
d3
```

Out[7]:

	a	b	c	d	e
0	2	3	4	6	8
1	4	5	6	20	24
2	1	2	3	2	3

In [ ]:

```
* We can delete a column using either del/pop
```

del

In [17]:

```
# deleting column 'd'
del d3['d']
d3
```

Out[17]:

	a	b	c	e
x	2	3	4	8
y	4	5	6	24
z	1	2	3	3

pop

In [13]:

```
# another way of deleting column 'a'
d3.pop('d')
d3
```

Out[13]:

	c	e
0	4	8
1	6	24
2	3	3

insert

In [14]:

```
# inserting a new column at 1 index
d3.insert(1,'new1',d3['c'])    #(loc,colname,data)
d3
```

Out[14]:

	c	new1	e
0	4	4	8
1	6	6	24
2	3	3	3

In [ ]:

```
* We can insert a new row using append function.
* dataframe.append() use deprecated in further versions of pandas. We can use 'pandas.conca
```

append()

In [18]:

```
# adding a row to existing DataFrame using append()
d={'c':1,'new1':33,'e':22}
df11=d3.append(d,ignore_index=True)
df11
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_6340\1553430484.py:3: Fu
tureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  df11=d3.append(d,ignore_index=True)
```

Out[18]:

	c	new1	e
0	4	4	8
1	6	6	24
2	3	3	3
3	1	33	22

accessing DataFrame columns and values

In [25]:

```
# creating a dataset
import numpy as np
d4=pd.DataFrame({'abc':np.random.randint(2,6,size=(10)),'bcd':np.random.randint(4,10,size=(
d4
```

Out[25]:

	abc	bcd	cde
0	5	4	9
1	5	7	8
2	3	8	5
3	3	6	8
4	3	5	5
5	2	5	6
6	4	5	9
7	5	6	5
8	3	6	6
9	3	6	5

In [20]:

```
# head() used to print first 5 rows
d4.head()
```

Out[20]:

	abc	bcd	cde
0	5	5	7
1	3	6	6
2	5	8	7
3	2	9	6
4	2	6	6

In [26]:

```
# we can print desired no of rows from 0 using head(no)
d4.head(8)
```

Out[26]:

	abc	bcd	cde
0	5	4	9
1	5	7	8
2	3	8	5
3	3	6	8
4	3	5	5
5	2	5	6
6	4	5	9
7	5	6	5

In [22]:

```
# tail() will return last 5 rows
d4.tail()
```

Out[22]:

	abc	bcd	cde
5	4	5	8
6	3	4	5
7	4	8	4
8	2	6	4
9	2	5	5

loc[ ]

In [24]:

```
# we can get our desired value using loc[row,column]
d4.loc[9,'cde']
```

Out[24]:

5

In [27]:

```
# geeting data from required row and columns
d4.loc[4:9,['abc','cde']]
```

Out[27]:

	abc	cde
4	3	5
5	2	6
6	4	9
7	5	5
8	3	6
9	3	5

In [28]:

```
# can get multiple rows and columns data per requirement
d4.loc[[3,4,7],['abc','cde']]
```

Out[28]:

	abc	cde
3	3	8
4	3	5
7	5	5

DataFrame info

In [23]:

```
# info() will give all the info about dataframe
d4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   abc     10 non-null     int32
 1   bcd     10 non-null     int32
 2   cde     10 non-null     int32
dtypes: int32(3)
memory usage: 248.0 bytes
```

iloc[ ]

In [29]:

```
# iloc will take row index and column index. loc take row name and column name
d4.iloc[9,2]
```

Out[29]:

```
5
```

In [31]:

```
# iloc to get required dataset
d4.iloc[2:7,[0,2]]
```

Out[31]:

	abc	cde
2	3	5
3	3	8
4	3	5
5	2	6
6	4	9

In [32]:

```
# dataframe.column.values will give dataset in array format
d4.abc.values
```

Out[32]:

```
array([5, 5, 3, 3, 3, 2, 4, 5, 3, 3])
```

In [33]:

```
# new column 'sub' will add to d4 data frame as a result of mentioned operations
d4['sub']=d4.abc.values-d4.cde.values-d4.bcd.values
d4
```

Out[33]:

	abc	bcd	cde	sub
0	5	4	9	-8
1	5	7	8	-10
2	3	8	5	-10
3	3	6	8	-11
4	3	5	5	-7
5	2	5	6	-9
6	4	5	9	-10
7	5	6	5	-6
8	3	6	6	-9
9	3	6	5	-8

In [34]:

```
# new data frame creation and assign column names through new df
a=[['rk',102,15000],['rama',103,20000],['krishna',104,25000]]
df1=pd.DataFrame(a,columns=['name','id','salary'])
df1
```

Out[34]:

	name	id	salary
0	rk	102	15000
1	rama	103	20000
2	krishna	104	25000

In [40]:

```
# condition checking in data frames
y=df1[df1.salary>=20000]
print(y)
y[['id','salary']]
```

```
      name   id  salary
1     rama  103   20000
2  krishna  104   25000
```

Out[40]:

	id	salary
1	103	20000
2	104	25000

In [41]:

```
# adding data to df using append
df1.append({'name':'sajan','id':105,'salary':30000},ignore_index=True)
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_6340\3954664748.py:2: Fu
tureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  df1.append({'name':'sajan','id':105,'salary':30000},ignore_index=True)
```

Out[41]:

	name	id	salary
0	rk	102	15000
1	rama	103	20000
2	krishna	104	25000
3	sajan	105	30000

nulls

In [42]:

```
# appending null value
df1=df1.append({'name':np.nan,'id':105,'salary':30000},ignore_index=True)
df1
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_6340\1195158177.py:1: Fu
tureWarning: The frame.append method is deprecated and will be removed from
pandas in a future version. Use pandas.concat instead.
  df1=df1.append({'name':np.nan,'id':105,'salary':30000},ignore_index=True)
```

Out[42]:

	name	id	salary
0	rk	102.0	15000.0
1	rama	103.0	20000.0
2	krishna	104.0	25000.0
3	NaN	105.0	30000.0

In [43]:

```
# isnull() will give 'True' if null value present in the data
df1.isnull()
```

Out[43]:

	name	id	salary
0	False	False	False
1	False	False	False
2	False	False	False
3	True	False	False

In [46]:

```
# gives total count of null values column wise
df1.isnull().sum()
```

Out[46]:

```
name      1
id        0
salary    0
dtype: int64
```

In [47]:

```
# dropping null value contained row
df1.dropna()
```

Out[47]:

	name	id	salary
0	rk	102.0	15000.0
1	rama	103.0	20000.0
2	krishna	104.0	25000.0

In [51]:

```
# changing null value to 'abc' to identify easily later
df1.fillna(value='abc')
```

Out[51]:

	name	id	salary
0	rk	102.0	15000.0
1	rama	103.0	20000.0
2	krishna	104.0	25000.0
3	abc	105.0	30000.0

groupby

In [55]:

```
# defining a dataset
df=pd.DataFrame({'Animal':['Falcon','Falcon','Parrot','Parrot'],'Max Speed':[380.0,370.0,24
df
```

Out[55]:

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

In [57]:

```
# Groupby using mean()
df.groupby(['Animal']).mean()
```

Out[57]:

Animal	Max Speed
Falcon	375.0
Parrot	25.0

csv reading

In [63]:

```
# reading already existing csv file into dataframe
import pandas as pd
df_csv=pd.read_csv('D:/DS Practice/KFC Menu.csv')
df_csv
```

Out[63]:

	Unnamed: 0	Item Name	Item Price
0	0	Biryani combo - for 2	795.24
1	1	10pc Strips with Dynamite Sauce	639.04
2	2	10pc Strips with Nashville Sauce	639.04
3	3	Classic Biryani Combo	740.00
4	4	Smoky Grilled Biryani Combo	740.00
...	...	...	...
142	142	Pepsi PET	57.14
143	143	Pepsi Can 330 ml	57.14
144	144	7UP Can 330 ml	57.14
145	145	Pepsi Black Can 330 ml	57.14
146	146	Mirinda Can 330 ml	57.14

147 rows × 3 columns

In [65]:

```
# checking null values in the above csv dataset
df_csv.columns
df_csv.isnull().sum()
```

Out[65]:

```
Unnamed: 0    0
Item Name     0
Item Price    0
dtype: int64
```

In [67]:

```
# getting values in array format
df_csv['Item Name'].values
```

Out[67]:

```
array(['Biryani combo - for 2', '10pc Strips with Dynamite Sauce',
       '10pc Strips with Nashville Sauce', 'Classic Biryani Combo',
       'Smoky Grilled Biryani Combo', 'Family Feast',
       '6pc Hot & Crispy with Dynamite Sauce', 'Bucket for Two',
       '6pc Hot & Crispy with Nashville Sauce',
       '5 pc Leg Piece Bucket Meal', 'Big 8', 'Ultimate Savings Bucket',
       '10pc Leg Piece Bucket & 4 Dips', '8 pcs Hot & Crispy Chicken',
       '6 pc Hot & Crispy Chicken', '5pc Leg Piece Bucket & 2 Dips',
       '5 pc Smoky Red Chicken', 'Classic Chicken Biryani Bucket-Large',
       'Popcorn Chicken Biryani Bucket-Large',
       'Smoky Grilled Biryani Bucket-Large',
       'Classic Chicken Biryani Bucket', 'Popcorn Chicken Biryani Bucket',
       'Smoky Grilled Biryani Bucket', 'Veg Biryani Bucket',
       'Buddy Popcorn Nachos Meal', 'Popcorn Nachos', 'Buddy Meal',
       'Chicken & Krispers Combo', 'Mixed Zinger Doubles',
       'Classic Zinger Meal', 'Tandoori Zinger Burger', 'Classic Zinger',
       '2 Chicken Krisper Burgers', '2 Veg Krisper Burgers',
       '2 Veg Krispers Meal', 'Veg Zinger Burger', 'Super Snacker Combo',
       "Chick'n Strips Solo Combo", 'Chicken & Fries Bucket',
       '4pcs Hot & Crispy Chicken', 'Biryani Rice',
       'Popcorn & Fries bucket', 'Large Popcorn', 'Medium Popcorn',
       'Regular Popcorn', '2 pc Hot & Crispy Chicken',
       '2 pcs Smoky Red Chicken', '6pcs Boneless Chicken strips',
       '1 pc Hot & Crispy', '1 pc Smoky Red Chicken',
       '3pc Boneless Chicken strips', '2 pc Veg Patty',
       'Spicy Mix Fries -Large', 'Spicy Mix Fries -Medium',
       'Nashville Hot Pepper Sauce Bottle', 'Coffee Mousse Cake',
       'Dynamite Spicy Mayo Sauce Bottle', 'Choco Mud Pie',
       'Tandoori Masala Dip', 'Pack of 4 Dips', 'Pack of 2 Dip',
       'Large Fries', 'Medium Fries', 'Pepsi PET', 'Pepsi Can 330 ml',
       '7UP Can 330 ml', 'Pepsi Black Can 330 ml', 'Mirinda Can 330 ml',
       'Biryani combo - for 2', 'House Party Combo - for Many',
       'Chick’n Dip Combo - for Many', '10pc Strips with Dynamite Sauce',
       '10pc Strips with Nashville Sauce', 'Classic Biryani Combo',
       'Smoky Grilled Biryani Combo', 'Family Feast',
       '6pc Hot & Crispy with Dynamite Sauce', 'Bucket for Two',
       '6pc Hot & Crispy with Nashville Sauce',
       'Friendship Bucket with Dynamite Sauce',
       'Friendship Bucket with Nashville Sauce', 'Stay Home Bucket',
       'Mingles Bucket Meal', '5 pc Leg Piece Bucket Meal', 'Big 8',
       'Ultimate Savings Bucket', '10pc Leg Piece Bucket & 4 Dips',
       '8 pcs Hot & Crispy Chicken', '6 pc Hot & Crispy Chicken',
       '5pc Leg Piece Bucket & 2 Dips', '5 pc Smoky Red Chicken',
       'Big 12 Bucket', 'Classic Chicken Biryani Bucket-Large',
       'Popcorn Chicken Biryani Bucket-Large',
       'Smoky Grilled Biryani Bucket-Large',
       'Classic Chicken Biryani Bucket', 'Popcorn Chicken Biryani Bucket',
       'Smoky Grilled Biryani Bucket', 'Veg Biryani Bucket',
       'Buddy Popcorn Nachos Meal', 'Popcorn Nachos',
       'Popcorn Nachos & Wings Combo', 'Buddy Meal',
       'Chicken & Krispers Combo', 'Mixed Zinger Doubles',
       'Classic Zinger Meal', 'Tandoori Zinger Burger', 'Classic Zinger',
       '2 Chicken Krisper Burgers', '2 Veg Krisper Burgers',
       '2 Veg Krispers Meal', 'Veg Zinger Burger', 'Super Snacker Combo',
```

```
       "Chick'n Strips Solo Combo", 'Chicken & Fries Bucket',
       '4pcs Hot & Crispy Chicken', 'Biryani Rice',
       'Popcorn & Fries bucket', 'Large Popcorn', 'Medium Popcorn',
       'Regular Popcorn', '2 pc Hot & Crispy Chicken',
       '2 pcs Smoky Red Chicken', '6pcs Boneless Chicken strips',
       '1 pc Hot & Crispy', '1 pc Smoky Red Chicken',
       '3pc Boneless Chicken strips', '2 pc Veg Patty', 'Mingles Bucket',
       'Chick’n Wings Combo', '4pc Hot Chicken Wings',
       'Spicy Mix Fries -Large', 'Spicy Mix Fries -Medium',
       'Nashville Hot Pepper Sauce Bottle', 'Coffee Mousse Cake',
       'Dynamite Spicy Mayo Sauce Bottle', 'Choco Mud Pie',
       'Tandoori Masala Dip', 'Pack of 4 Dips', 'Pack of 2 Dip',
       'Large Fries', 'Medium Fries', 'Pepsi PET', 'Pepsi Can 330 ml',
       '7UP Can 330 ml', 'Pepsi Black Can 330 ml', 'Mirinda Can 330 ml'],
      dtype=object)
```

column/row wise iterations

In [70]:

```
# looping data column wise using iteritems()
import pandas as pd
import numpy as np
df=pd.DataFrame(np.random.randn(4,3),columns=['col1','col2','col3'])
print(df)
print('\n')
for key,value in df.iteritems():
    print(key,value)
    print('\n')
```

```
       col1      col2      col3
0 -0.771835 -0.755471 -0.531186
1 -0.380843 -1.571319  0.463389
2  0.905633  0.331179  0.122188
3  1.180244 -1.510906  1.526463


col1 0   -0.771835
1   -0.380843
2    0.905633
3    1.180244
Name: col1, dtype: float64


col2 0   -0.755471
1   -1.571319
2    0.331179
3   -1.510906
Name: col2, dtype: float64


col3 0   -0.531186
1    0.463389
2    0.122188
3    1.526463
Name: col3, dtype: float64
```

In [72]:

```
# looping table row wise use iterrows()
import pandas as pd
import numpy as np

df=pd.DataFrame(np.random.randn(3,3),columns=['col1','col2','col3'])
print(df)
print('\n')
for row in df.iterrows():
    print(row)
    print('\n')
```

```
       col1      col2      col3
0  1.797755 -1.701335 -0.853244
1 -1.339501  1.130570  0.991683
2  0.499060  0.224174 -1.926962


(0, col1    1.797755
col2   -1.701335
col3   -0.853244
Name: 0, dtype: float64)


(1, col1   -1.339501
col2    1.130570
col3    0.991683
Name: 1, dtype: float64)


(2, col1    0.499060
col2    0.224174
col3   -1.926962
Name: 2, dtype: float64)
```

Transpose

In [74]:

```
# Transpose
df.T
```

Out[74]:

	0	1	2
col1	1.797755	-1.339501	0.499060
col2	-1.701335	1.130570	0.224174
col3	-0.853244	0.991683	-1.926962

In [77]:

```
# defining dataset
import pandas as pd
data={'firstname':['Arun','Jebu','Venkat','Rekha','Majid','Mohsin'],
     'lastname':['Kumar','Jacob','Raghavan','Singh','Khan','Khan'],
     'employmenttype':['Service','Business','Student','Service','Business','Business'],
     'country':['India','USA','USA','Sweden','Australia','Germany']}
df=pd.DataFrame(data,columns=['firstname','lastname','employmenttype','country'])
print(df)
```

```
  firstname  lastname employmenttype    country
0      Arun     Kumar        Service      India
1      Jebu     Jacob       Business        USA
2    Venkat  Raghavan        Student        USA
3     Rekha     Singh        Service     Sweden
4     Majid      Khan       Business  Australia
5    Mohsin      Khan       Business    Germany
```

In [79]:

```
# if both employmenttype and country matches, it will return 1 else 0.
# These tables are used in sports manily
df1=pd.get_dummies(df['employmenttype'])
df2=pd.get_dummies(df['country'])
print(df1)
print(df2)
```

```
   Business  Service  Student
0         0        1        0
1         1        0        0
2         0        0        1
3         0        1        0
4         1        0        0
5         1        0        0
   Australia  Germany  India  Sweden  USA
0          0        0      1       0    0
1          0        0      0       0    1
2          0        0      0       0    1
3          0        0      0       1    0
4          1        0      0       0    0
5          0        1      0       0    0
```

In [81]:

```
# concating datasets. we can remove emptype and country and can use other details if needed
frames=[df,df1,df2]
result=pd.concat(frames,axis=1)
result
```

Out[81]:

	firstname	lastname	employmenttype	country	Business	Service	Student	Australia	Gern
0	Arun	Kumar	Service	India	0	1	0	0	
1	Jebu	Jacob	Business	USA	1	0	0	0	
2	Venkat	Raghavan	Student	USA	0	0	1	0	
3	Rekha	Singh	Service	Sweden	0	1	0	0	
4	Majid	Khan	Business	Australia	1	0	0	1	
5	Mohsin	Khan	Business	Germany	1	0	0	0	

In [82]:

```
# defining indian cities weather
import pandas as pd
india_weather=pd.DataFrame({'city':['mumbai','delhi','banglore'],
                            'temperature':[32,45,30],
                            'humidity':[80,60,78]})
india_weather
```

Out[82]:

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78

In [84]:

```
# defining foreign cities weather
us_weather=pd.DataFrame({'city':['new york','chicago','orlando'],
                            'temperature':[21,14,35],
                            'humidity':[68,65,75]})
us_weather
```

Out[84]:

	city	temperature	humidity
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

index variations

In [86]:

```
# index will repeat if normally concats.
df=pd.concat([india_weather,us_weather])
df
```

Out[86]:

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78
0	new york	21	68
1	chicago	14	65
2	orlando	35	75

In [87]:

```
# ignore index to continue index from previous onwards
df=pd.concat([india_weather,us_weather],ignore_index=True)
df
```

Out[87]:

	city	temperature	humidity
0	mumbai	32	80
1	delhi	45	60
2	banglore	30	78
3	new york	21	68
4	chicago	14	65
5	orlando	35	75

In [89]:

```
# concatination using keys
df=pd.concat([india_weather,us_weather],keys=['India','US'])
df
```

Out[89]:

		city	temperature	humidity
	0	mumbai	32	80
India	**1**	delhi	45	60
	2	banglore	30	78
	0	new york	21	68
US	**1**	chicago	14	65
	2	orlando	35	75

Relationships

In [ ]:

```
* A great aspect of the Pandas module is the corr() method.

* The corr() method calculates the relationship between each column in your data set.

* df.corr() - correlation is used to check how strong relationship between two columns.

* we have 3 types of corelations.

    * perfect correlation

        We can see that "Duration" and "Duration" got the number 1.000000,
        which makes sense, each column always heas a perfect relationship with itself.

    * good correlation

        "Duration" and "Calories" got a 0.922721 correlation, which is a very good correlat
        and we can predict that the longer you work out, the more calories you burn, and th
        if you burned a lot of calories, you probably had a long work out.

    * bad correlation

        "Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlati
        meaning that we can not predict the max pulse by just looking at the duration of th
```

In [3]:

```
# inserting image into jupyter
from IPython.display import Image
Image(filename="D:/Courses/CodigGrad/images/correlation.png",width=400,height=400)
# Below is the image
```

Out[3]:


```
df=pd.read_csv('data_panda.csv')
df.corr()
```

	Duration	Pulse	Maxpulse	Calories
Duration	1.000000	-0.155408	0.009403	0.922717
Pulse	-0.155408	1.000000	0.786535	0.025121
Maxpulse	0.009403	0.786535	1.000000	0.203813
Calories	0.922717	0.025121	0.203813	1.000000

Correlation

In [ ]:

```
df=pd.read_csv('data_panda.csv')
df.corr()
```

drop_duplicates()

In [92]:

```
# deleting duplicates using drop_duplicates()
import pandas as pd
emp={'Name':['Parker','Smith','William','Parker'],
    'Age':[21,32,29,21]}
info=pd.DataFrame(emp)
print(info)
print('----------------')
info=info.drop_duplicates()
print(info)
```

```
      Name  Age
0   Parker   21
1    Smith   32
2  William   29
3   Parker   21
----------------
      Name  Age
0   Parker   21
1    Smith   32
2  William   29
```

In [93]:

```
# writing function to work on any dataset
import pandas as pd
def calc_sum(x):
    return x+1

data={
    'x':[50,40,30],
    'y':[300,1112,42]
}
df=pd.DataFrame(data)
print(df)
```

```
    x     y
0  50   300
1  40  1112
2  30    42
```

apply()

In [95]:

```
# we can take any dataset and apply above function using apply()
x=df.apply(calc_sum)
print(x)
```

```
    x     y
0  51   301
1  41  1113
2  31    43
```

joins

In [98]:

```
# explore sql joins
import pandas as pd
data1={
    'name':['Sally','Mary','John'],
    'age':[50,40,30]
}
data2={
    'name':['Sally','Peter','Micky'],
    'age':[77,44,22]
}

df1=pd.DataFrame(data1)
df2=pd.DataFrame(data2)
print(df1)
print('----------')
print(df2)
print('----------')
newdf=df1.merge(df2,how='right')
print(newdf)
```

```
    name  age
0  Sally   50
1   Mary   40
2   John   30
----------
    name  age
0  Sally   77
1  Peter   44
2  Micky   22
----------
    name  age
0  Sally   77
1  Peter   44
2  Micky   22
```

In [99]:

```
#outer join
df3=pd.merge(df1,df2,on='name',how='outer')
df3
```

Out[99]:

	name	age_x	age_y
0	Sally	50.0	77.0
1	Mary	40.0	NaN
2	John	30.0	NaN
3	Peter	NaN	44.0
4	Micky	NaN	22.0

In [100]:

```
#left join
df3=pd.merge(df1,df2,on='name',how='left')
df3
```

Out[100]:

	name	age_x	age_y
0	Sally	50	77.0
1	Mary	40	NaN
2	John	30	NaN

In [102]:

```
#right join
df3=pd.merge(df1,df2,on='name',how='right')
df3
```

Out[102]:

	name	age_x	age_y
0	Sally	50.0	77
1	Peter	NaN	44
2	Micky	NaN	22

In [103]:

```
#inner join
df3=pd.merge(df1,df2,on='name',how='inner')
df3
```

Out[103]:

	name	age_x	age_y
0	Sally	50	77

In [ ]:

```
'''
Task - 40
Practice total pandas
Task - 41
Prepare a jupyter notebook on pandas
Task - 42
Complete total 300 interview questions
'''
```

In [ ]:

```
!pip install matplotlib
from matplotlib import pyplot as plt
import numpy as np
```

Data Visualization

In [ ]:

```
* Data Visualization is the graphical representation of data to help people understand
  context and significance.

* Interactive data Visualization enables companies to drill down to explore details,
  identify patterns and outliers.

* DV Libraries: matplotlib, seaborn, ggplot2, Bokeh, plotly

* DV Tools: PowerBI, tableau, Qlik.

* Tools are user friendly when compared to Libraries.

* Libraries used for small amount of data while tools for larger amount of data.
```

In [30]:

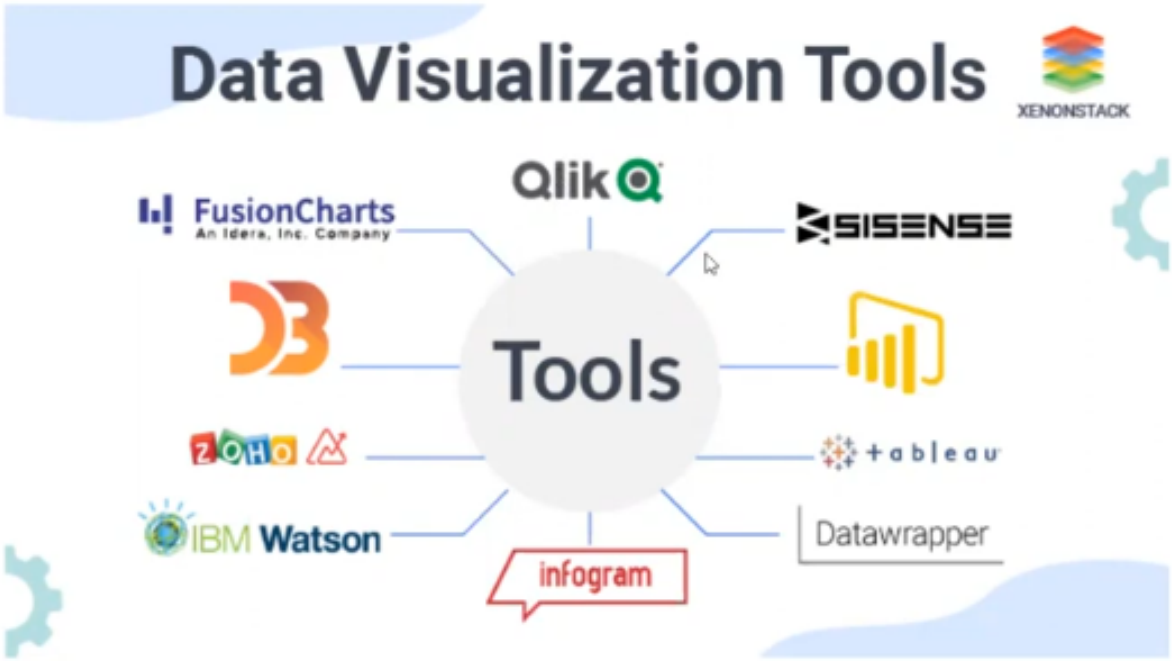
```
# from IPython.display import Image
# Image(filename="D:/Courses/CodigGrad/images/data_visualization_libraries.png",width=600,
```

Out[30]:

In [27]:

```
# Image(filename="D:/Courses/CodigGrad/images/data_visualization_tools.png",width=600, heig
```

Out[27]:



matplotlib

In [ ]:

```
* matplotlib is a one of the most popular Data Visualization Library.

* installation: !pip install matplotlib
                from matplotlib import pyplot as plt
                # import numpy as np    (# We can import other libraries based on the requi
```

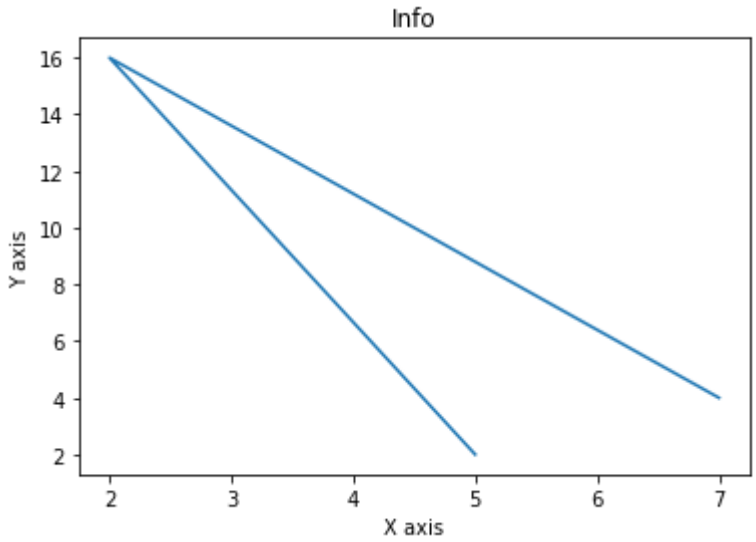
plotting data

In [ ]:

```
* Plotting will be done as (x,y) co-ordinates type.

* plt.title() - will give title to our plot
* plt.xlabel() - will give name to x axis
* plt.ylabel() - will give name to y axis
* plt.show() - to show plotted data (its like print())
```

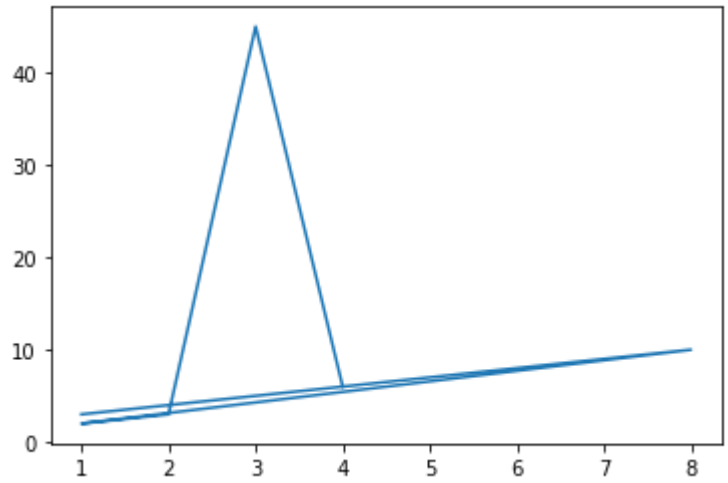
In [2]:

```
# plotting using list data
x=[5,2,7]
y=[2,16,4]        # co-ordinates (x,y)=(5,2) (2,16) (7,4)
plt.plot(x,y)
plt.title('Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```



In [3]:

```
# plotting using array data

#import numpy as np
# as we are using np here numpy need to import first
xpoints=np.array([1,8,1,2,3,4])
ypoints=np.array([3,10,2,3,45,6])     #(x,y) cordinates

plt.plot(xpoints,ypoints)
plt.show()
```



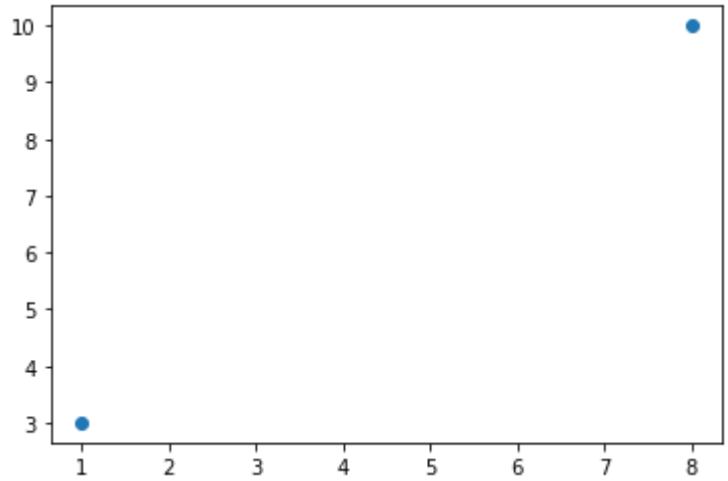
plotting without line

In [ ]:

* we can use 'o' in plot() to get plotting without lines.

In [4]:

```
xpoints=np.array([1,8])
ypoints=np.array([3,10])

plt.plot(xpoints,ypoints,'o')
plt.show()
```



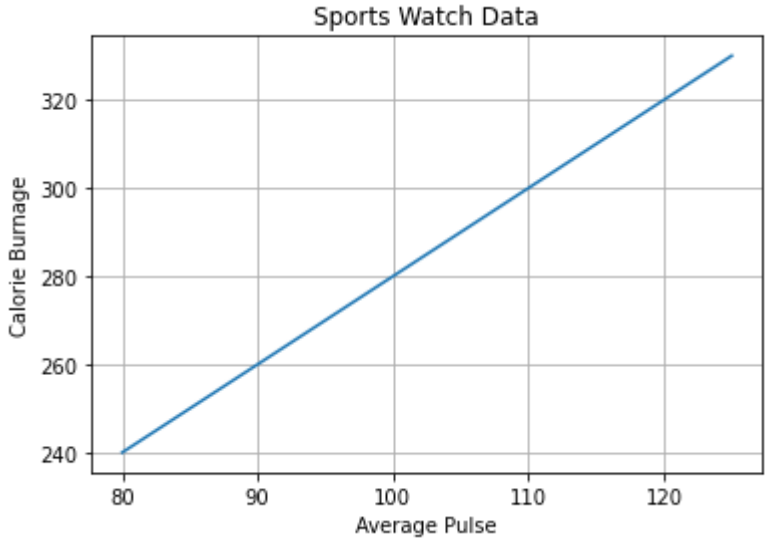
plotting with grid lines

In [ ]:

* with pyplot, we can use the grid() function to add grid lines to the plot

In [6]:

```
x=np.array([80,85,90,95,100,105,110,115,120,125])
y=np.array([240,250,260,270,280,290,300,310,320,330])

plt.title('Sports Watch Data')
plt.xlabel('Average Pulse')
plt.ylabel('Calorie Burnage')

plt.plot(x,y)
plt.grid()
plt.show()
```



Markers

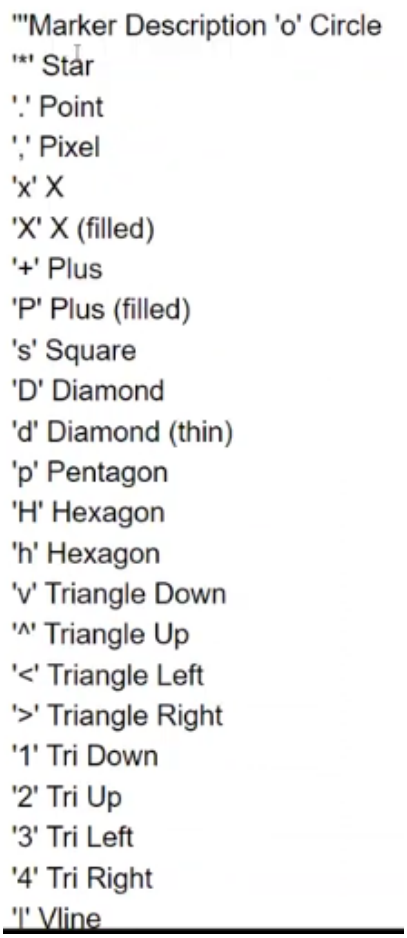
In [ ]:

```
* markers used to reshape (x,y) co-ordinate points as per our choice
  (dot, hexagon, diamon, etc).
* Based on the marker='' point type will change. Below image contains all
  pointer types.
```

In [33]:

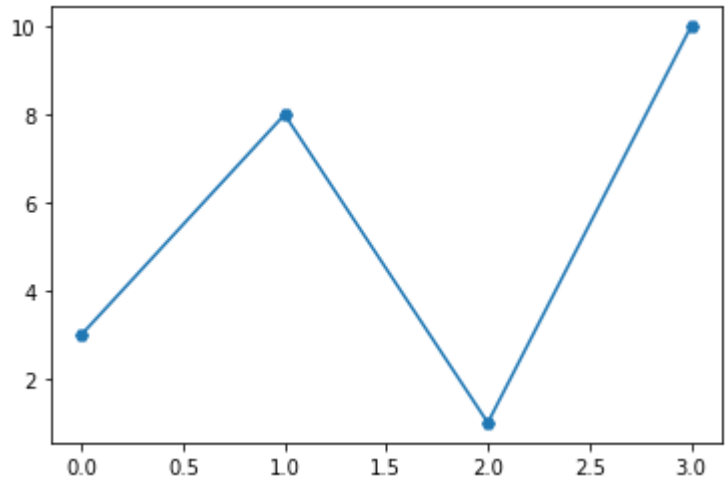
```
* Here are the list of marker types
# Image(filename="D:/Courses/CodigGrad/images/matplotlib_markers.png",width=200, height=400
```

Out[33]:



In [8]:

```
ypoints=np.array([3,8,1,10])
#plt.plot(ypoints,marker='o')
plt.plot(ypoints,marker='H')
plt.show()
```



In [ ]:

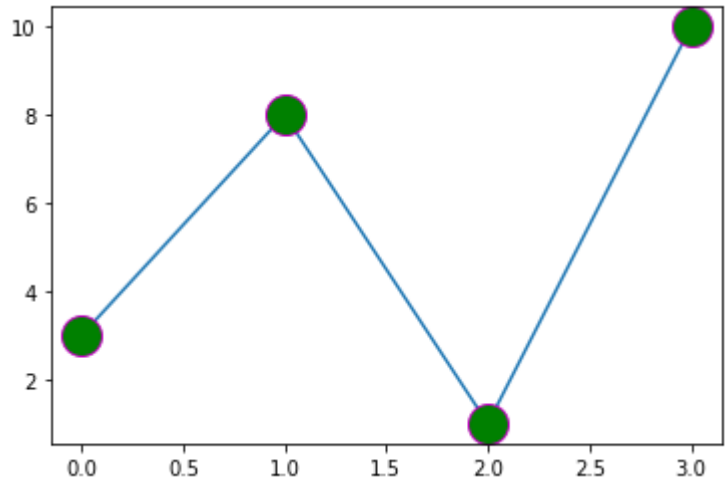
```
* We can change colors and sizes of the marker per requirement.

* ms - marker size
* mec - marker edge color
* mfc - marker face color
```

In [34]:

```
plt.plot(ypoints,marker='o', ms=20, mec='m', mfc='g') # edge color # face color
```

Out[34]:

```
[<matplotlib.lines.Line2D at 0x1fa18a389a0>]
```



In [35]:

```
* Here are the list of marker colors
# Image(filename="D:/Courses/CodigGrad/images/marker_colors.png",width=200, height=400)
```

Out[35]:

Color Syntax Description 'r' Red
'g' Green
'b' Blue
'c' Cyan
'm' Magenta
'y' Yellow
'k' Black
'w' White

In [36]:

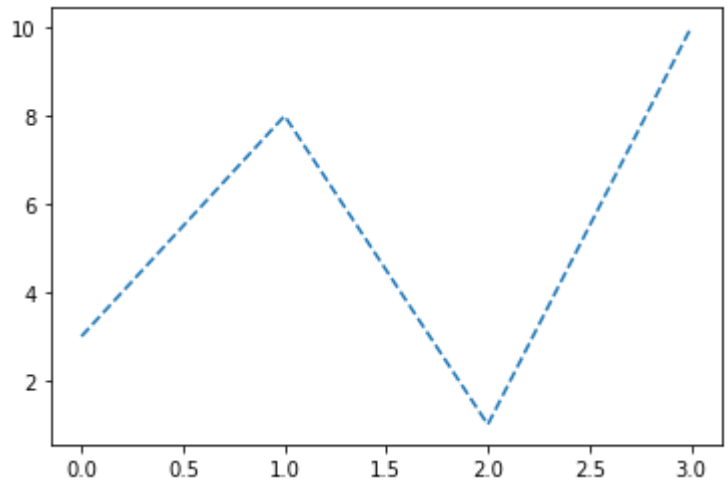
```
* We can change type of line also as below.
# Image(filename="D:/Courses/CodigGrad/images/lines_plot.png",width=200, height=400)
```

Out[36]:

'solid' (default) '-'
'dotted' ':'
'dashed' '--'
'dashdot' '-.'
'None' '' or ' '

In [38]:

```
ypoints = np.array([3,8,1,10])

plt.plot(ypoints, linestyle = 'dashed')
plt.show()
```

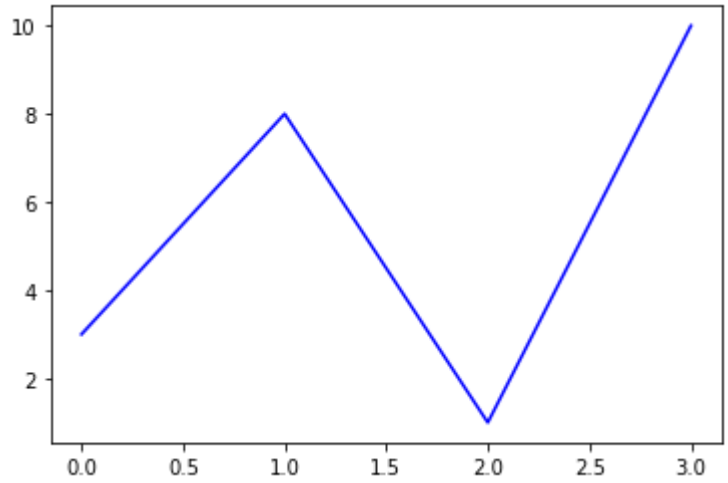


In [ ]:

```
* We can change line color also.
```

In [39]:

```
plt.plot(ypoints, color = 'b')
plt.show()
```

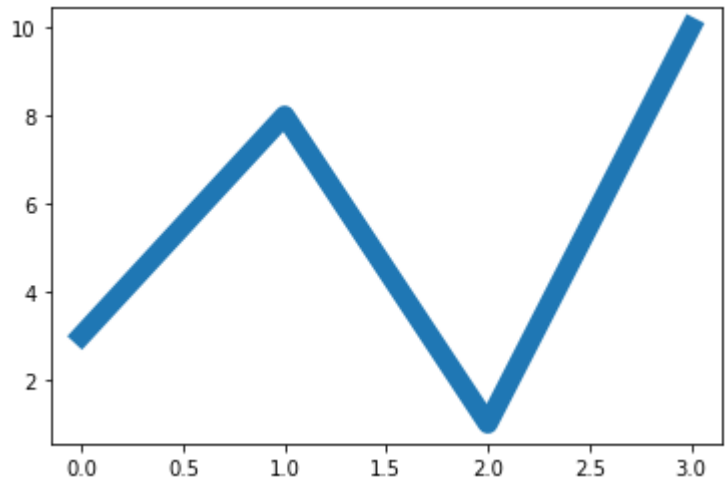


In [ ]:

```
* We can change line width also.
```

In [40]:

```
plt.plot(ypoints, linewidth='10')
plt.show()
```



subplot

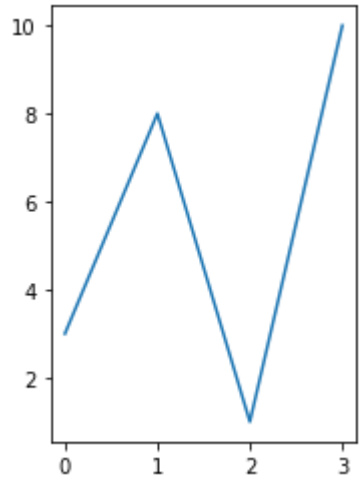
In [ ]:

```
* Subplot is nothing but Display Multiple Plots
* With the subplot() function, we can draw multiple plots in one figure
* subplot() function will take 3 parameters (rows, columns, positions)
  Eg: (2,3,6)
           * 6 is the maximum subplots here because (2,3,6) is (2x3=6)
```

In [43]:

```
# subplot() 1:
x=np.array([0,1,2,3])
y=np.array([3,8,1,10])
print(x)
print(y)
plt.subplot(1,2,1)
plt.plot(x,y)
plt.show()
```

```
[0 1 2 3]
[ 3  8  1 10]
```

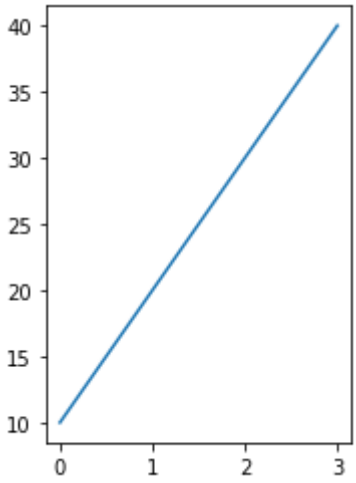


In [48]:

```
# subplot() 2:
x=np.array([0,1,2,3])
y=np.array([10,20,30,40])

plt.subplot(1,2,2)
plt.plot(x,y)
```

Out[48]:

```
[<matplotlib.lines.Line2D at 0x1fa18620e50>]
```

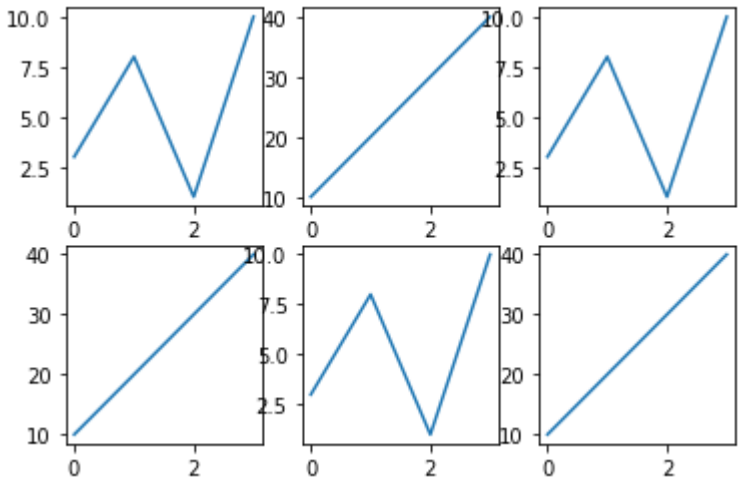


In [49]:

```
# 6 subplots in single figure
x=np.array([0,1,2,3])
y=np.array([3,8,1,10])
plt.subplot(2,3,1)
plt.plot(x,y)

x=np.array([0,1,2,3])
y=np.array([10,20,30,40])
plt.subplot(2,3,2)
plt.plot(x,y)

x=np.array([0,1,2,3])
y=np.array([3,8,1,10])
plt.subplot(2,3,3)
plt.plot(x,y)

x=np.array([0,1,2,3])
y=np.array([10,20,30,40])
plt.subplot(2,3,4)
plt.plot(x,y)

x=np.array([0,1,2,3])
y=np.array([3,8,1,10])
plt.subplot(2,3,5)
plt.plot(x,y)

x=np.array([0,1,2,3])
y=np.array([10,20,30,40])
plt.subplot(2,3,6)
plt.plot(x,y)
```

Out[49]:

```
[<matplotlib.lines.Line2D at 0x1fa18b04b50>]
```



Types of Data

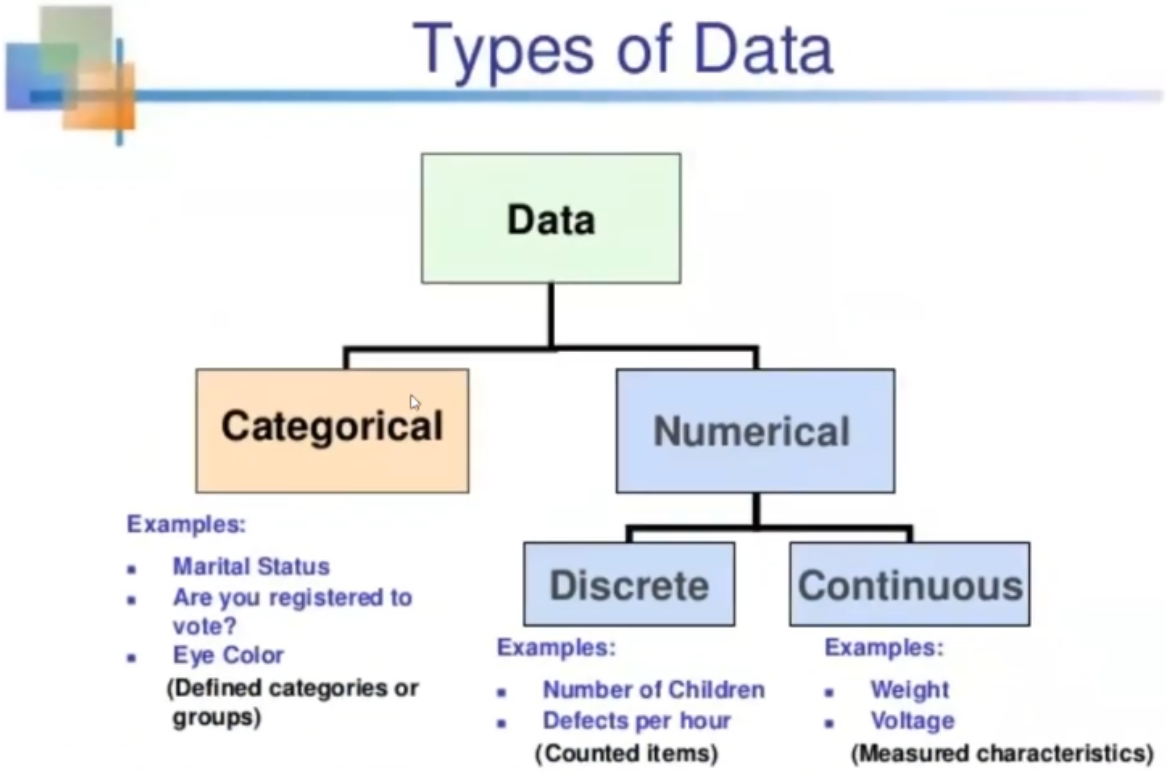
In [ ]:

```
* Data is nothing but the output of converted raw information.
* Data can be divided into 2 types.
     (i) Categorical              (ii) Numerical (Discrete, Continuous)
* Defined categories or groups like Marital status, Eye color are
  categorical data type.
* Countable data (no of children, defects per hour) come under Discrete
  data.
* Measured Characterstics(Weight, Voltage, stock market, population) type
  of data is continuous.
```

In [51]:

```
# Image(filename="D:/Courses/CodigGrad/images/types_of_data.png",width=600, height=400)
```

Out[51]:



Bar plot

In [ ]:

```
* Bar graphs are one of the most common types of graphs and are used to show data
  associated with the categorical variables. Matplotlib provides a bar() to make bar
  graphs which accepts arguments such as: categorical variables, their value and color.
```

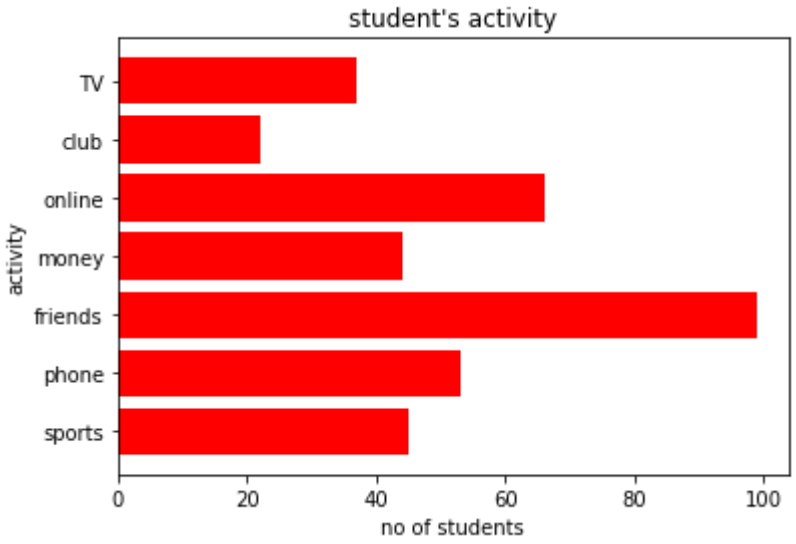
Horizontal Bar Chart

In [ ]:

```
* barh - used to plot horizontal bar chart
* bar - used to plot vertical bar chart

* There will be gap between each bar. Each bar of same size in width
```

In [54]:

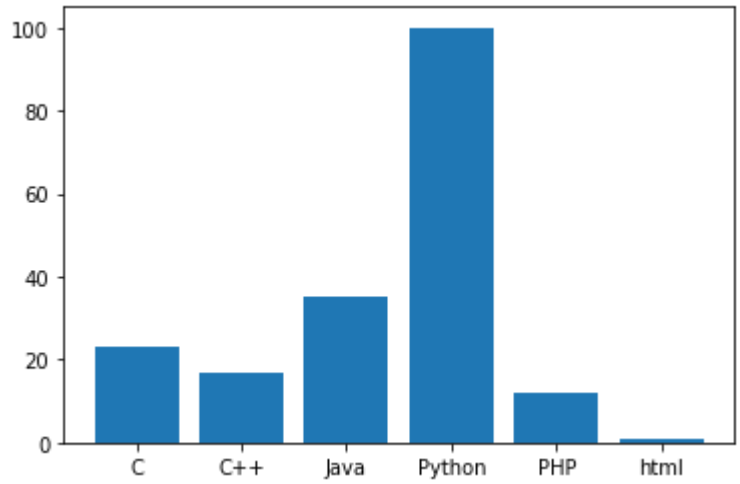
```
activities=["sports","phone","friends","money","online","club","TV"]
frequency=[45,53,99,44,66,22,37]

plt.barh(activities, frequency, color='r') # default colour is blue, color is used to chang
plt.title("student's activity")

plt.xlabel("no of students")
plt.ylabel("activity")

plt.show()
```



Vertical Bar Chart

In [55]:

```
langs = ['C','C++','Java','Python','PHP','html']
students = [23,17,35,100,12,1]
plt.bar(langs,students)
plt.show()
```

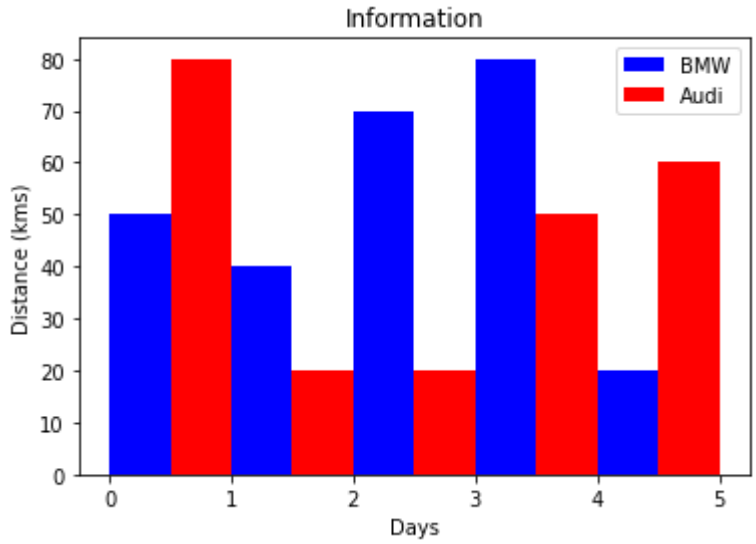


In [ ]:

```
* legend - Its used to summarize the data for more readability
* label(BMW) - this will show only legend() used.
```

In [58]:

```
plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],
label="BMW",color='b',width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],
label="Audi",color='r',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```

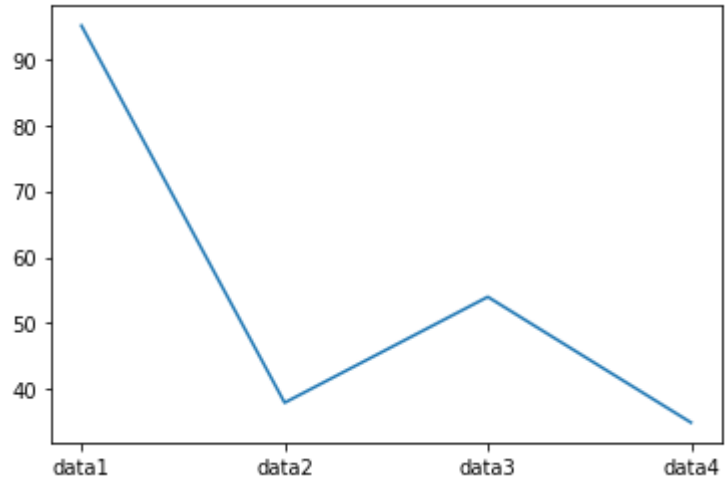


In [ ]:

```
* We can change the default scale on x and y axis using xticks/yticks
* labels will replace the scale on x axis if we use xticks(x,labels).
```

In [59]:

```
x=[1,2,3,4]
y=[95,38,54,35]
labels=['data1','data2','data3','data4']

plt.plot(x,y)
plt.xticks(x,labels)
plt.show()
```



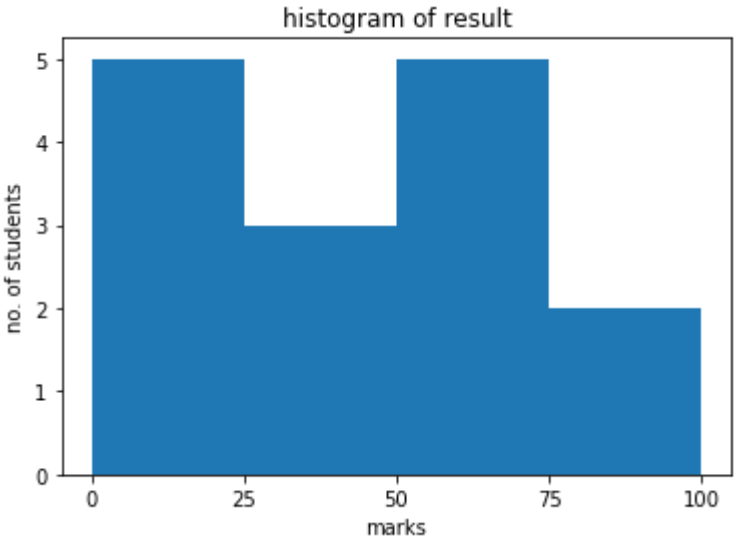
Histogram plot

In [ ]:

```
* First, we need to understand the difference between the bar graph and histogram.
  A histogram is used for the distribution, whereas a bar chart is used to compare
  different entities. A histogram is a type of bar plot that shows the frequency of a
  number of values compared to a set of values ranges.

* For example we take the data of the different age group of the people and plot a
  histogram with respect to the bin. Now, bin represents the range of values that
  are divided into series of intervals. Bins are generally created of the same size.

* In bar charts, we call them as bars. In histogram, we call them as bins.
* bars will have same width but bins are not in same width.
* There will be no gap betwwen bins because of continuous data flow but we have gaps
  between bars.
```

In [60]:

```
fig,ax = plt.subplots(1,1)
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
ax.hist(a, bins = [0,25,50,75,100])
ax.set_title("histogram of result")
ax.set_xticks([0,25,50,75,100])
ax.set_xlabel('marks')
ax.set_ylabel('no. of students')
plt.show()
```



Bar Chart vs Histogram

In [ ]:

```
* There are gaps between bars vs There are no gaps between bins and are different
  widths
* Bars: x-axis can have Words or Categories but Y-axis have no of people or frequency
  in Bar Chart
* Histogram: x- axis have continuous data and can take frequency in y.
```

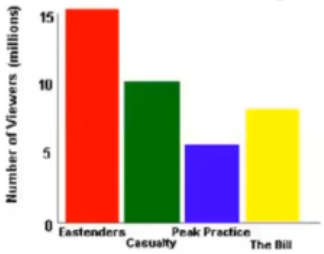
In [61]:

```
# Image(filename="D:/Courses/CodigGrad/images/bar_vs_histogram.png",width=600, height=400)
```

Out[61]:

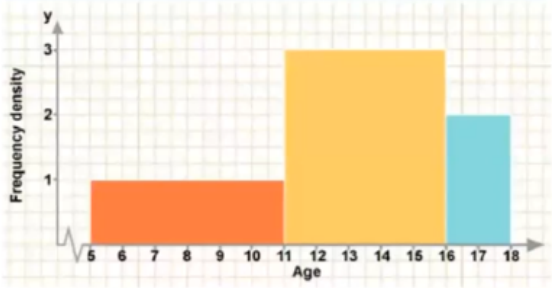
Bar Chart

Category	Property
Bars	There are gaps between the bars.
X-Axis	Words or categories. (Discrete)
Y-Axis	Number of people or frequency.



Histogram

Category	Property
Bars	There are no gaps between the bars and they are different widths.
X-Axis	Numbers. (Continuous)
Y-Axis	Frequency density – what's that?



pie plot

In [ ]:

```
* A pie chart is a circular graph that is broken down in the segment or slices of pie.
  It is generally used to represent the percentage or proportional data where each slice
  of pie represents a particular category.

* labels - names of the parts
* explode - used to separate certain part from the total pie. We can notice little bit
  separation at China & Pakistan parts.
* shadow - gives shadow effect. We can notice this at Pakistan part.
* startangle - We can put anything like 100/180/accute angel/right angle
* wedgeprops - These are the list of properties applied to pie
* edgecolor - It gives mentioned color to the edges.
* linewidth - This is used to increase/decrease edge line width
* colors - We can take direct color name or hexa codes using color picker from google.
```

In [84]:

```
data = [1433783686, 1366417754, 329064917, 270625568, 216565318]
l = ['China', 'India', 'Unites States', 'Indonesia', 'Pakistan',]
e = [0.1, 0.0, 0.0, 0.0, 0.1]
colors = ['#ccffcc', 'orange', 'blue', 'green', '#333300']

plt.pie(data, labels=l, explode=e, shadow=True, startangle=90,
       wedgeprops={'edgecolor':'black', 'linewidth': 2 }, colors=colors)

plt.title("Population Pie Chart")
plt.show()
```

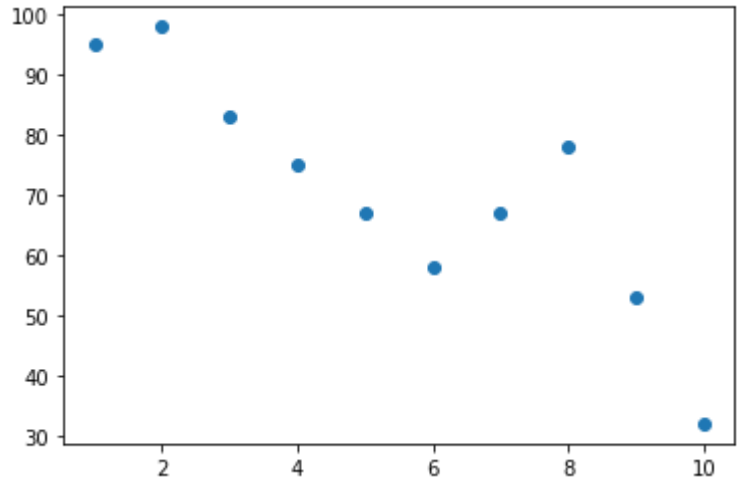
Scatter plot

In [ ]:

```
* The scatter plots are mostly used for comparing variables when we need to define how
  much one variable is affected by another variable. The data is displayed as a collection
  of points. Each point has the value of one variable, which defines the position on the
  horizontal axes, and the value of other variable represents the position on the vertical
  axis.

* Scatter plots are used to know how two variables matching.
```

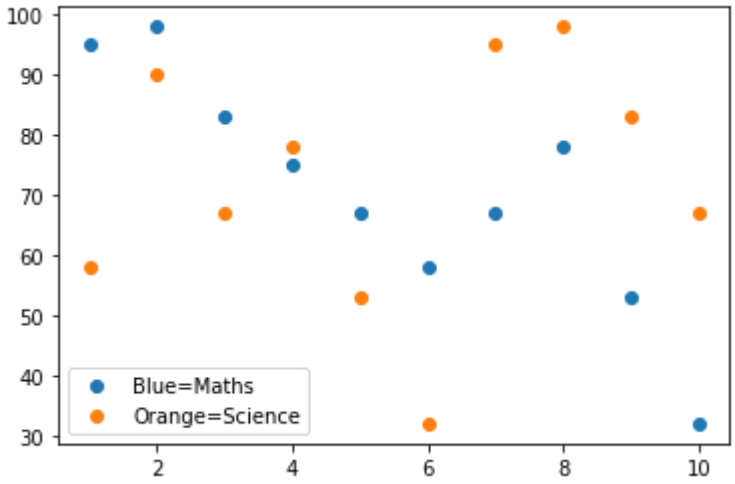
In [3]:

```
#datasets
students_id=[1,2,3,4,5,6,7,8,9,10]
student_marks=[95,98,83,75,67,58,67,78,53,32]
#scatter plot for the dataset
plt.scatter(students_id, student_marks)
plt.show()
```



In [5]:

```
# Maths Marks
students_id = np.array([1,2,3,4,5,6,7,8,9,10])
students_marks = np.array([95,98,83,75,67,58,67,78,53,32])
plt.scatter(students_id, students_marks,label='Blue=Maths')

#Science marks
students_id = np.array([1,2,3,4,5,6,7,8,9,10])
students_marks = np.array([58,90,67,78,53,32,95,98,83,67,])
plt.scatter(students_id, students_marks,label='Orange=Science')
plt.legend()
plt.show()
```



violin plot

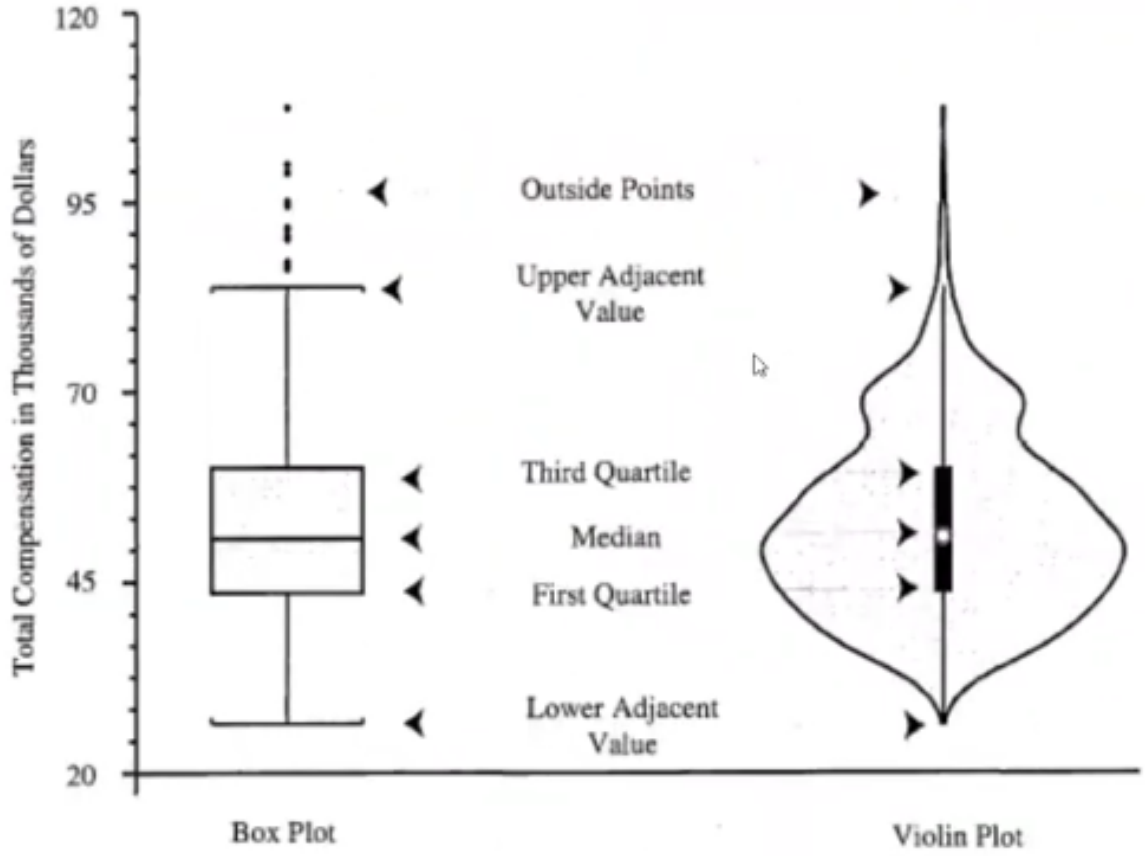
In [ ]:

```
* These plots are mainly a combination of Box Plots and Histograms.
* The violin plot usually portrays the distribution, median, interquartile range of data.
* violin plot is used to find outliers in the dataset.
* This can be clearly understand after Stats class.
* This is used to find out outliers. Generally 10th students are of age group 15-16. If
  anyone have 25/10 years, they are the outliers. This means extreme high/ extreme lows
  are outliers.
```

In [7]:

```
# Image(filename="D:/Courses/CodigGrad/images/violin_plot.png",width=600, height=400)
```

Out[7]:



Box Plot

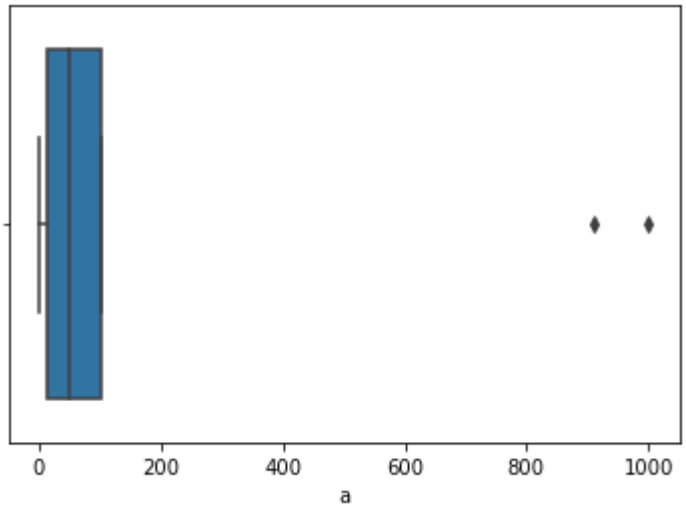
In [9]:

```
import seaborn as s
import pandas as pd
l=[0,1,50,60,50,14,909,1000,101]
df=pd.DataFrame(l,columns=['a'])
df
s.boxplot(df['a'])
# ending diamond symbols are outliers in the data
```

```
C:\Users\Administrator\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

Out[9]:

```
<AxesSubplot:xlabel='a'>
```



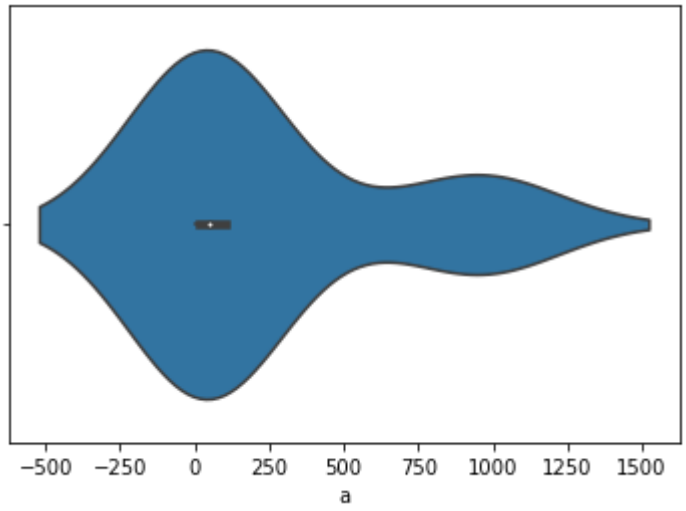
In [10]:

```
s.violinplot(df['a'])
# We can understand this after Stats class
```

```
C:\Users\Administrator\anaconda3\lib\site-packages\seaborn\_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passing oth
er arguments without an explicit keyword will result in an error or misinter
pretation.
  warnings.warn(
```

Out[10]:

```
<AxesSubplot:xlabel='a'>
```

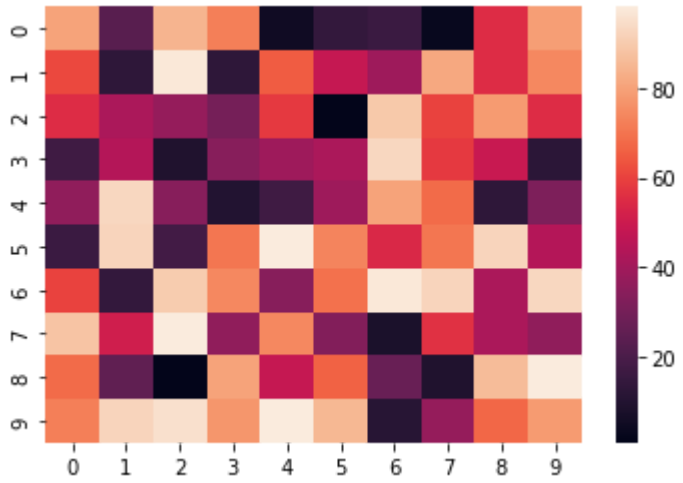


heatmap

In [11]:

```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data=np.random.randint(low=1,high=100,size=(10,10))
print("The data to be plotted: \n")
print(data)

# plotting the heatmap
hm = sn.heatmap(data = data)

# displaying the plotted heatmap
plt.show()
```

```
The data to be plotted:

[[80 23 84 72  5 14 16  3 55 79]
 [61 13 97 13 65 48 39 81 55 74]
 [55 42 37 30 58  1 89 60 78 55]
 [17 44  9 34 39 42 93 58 49 12]
 [36 93 34 10 17 39 80 68 13 32]
 [16 92 18 70 98 73 54 70 92 44]
 [60 14 90 74 34 69 97 92 42 93]
 [88 51 98 36 74 33  8 56 42 36]
 [68 25  1 80 48 66 27  9 86 98]
 [72 92 95 77 98 85 11 37 67 78]]
```



EDA Process

In [ ]:

```
* ipynb - interactive python notebook
* EDA Process - Explorative Data Analysis
    Data Gathering
    Data Preprocessing
    Data Visualization
```

In [ ]:

```
Step-1: Check data whether its clear or messy with null values and unwanted columns
Step-2: Check info(), check head(), check tail()
Step-3: Remove unwanted columns as below. 'unnamed: 9' are column names contains NaN
        values. inplace=True will permanently delte columns
        s=data.drop(["Unnamed: 7","Unnamed: 9","Unnamed: 10"],axis='columns',inplace=True)
        #remove or del col
Step-4: Check for any null values using data.isnull().sum()
Step-5: Remove null value contained rows (based on data set its decided can we delete
        entire rows or not)
Step-6: check if any spelling mistakes and correct it like below.
        data=data.replace('Canda','Canada',regex=True)  # As we are operating on strings,
        #put regex=True
        data.head()
        data.dtypes
Step-7: As we checked, all data in string(object) type but money should be in float. We can
        remove $ symbols as below.
        data=data.replace(['\$'," "],"",regex=True)   # Removes $ symbol before money
Step-8: Convert K, M(Milliion), B(Billion) into rupees using below function.
        def value_to_float(x):
            if 'K' in x:
                if len(x) > 1:
                    return float(x.replace('K','')) * 0.000001
            if 'M' in x:
                if len(x) > 1:
                    return float(x.replace('M','')) * 1000
            if 'B' in x:
                if len(x) > 1:
                    return float(x.replace('B','')) * 100000
        data['$ Last Change'] = data['$ Last Change'].apply(value_to_float)
Step-9: Send data into various charts finally.
```

In [30]:

```
#Image(filename="D:/Courses/CodigGrad/images/eda.png",width=600, height=400)
```

Out[30]:

	Rank	Name	Total Net Worth	$ Last Change	$ YTD Change	Country	Industry
0	1.0	Elon Musk	$270B	-$2.89B	+$773M	United States	Technology
1	2.0	Jeff Bezos	$188B	+$1.68B	-$2.31B	United States	Technology
2	3.0	Bernard Arnault	$155B	+$892M	+$40.9B	France	Consumer
3	4.0	Bill Gates	$144B	-$1.32B	+$12.2B	United States	Technology
4	5.0	Mark Zuckerberg	$114B	+$203M	+$10.9B	United States	Technology

In [18]:

```
import numpy as np    # linear algebra
import pandas as pd   # data processing, CSV file I/O
```

In [ ]:

```
* use engine='python' - To overcome below error. default engine is C.
    ParserError: Error tokenizing data. C error: EOF inside string starting at row 1001
* error_bad_lines=False - To overcome below error.
    unexpected end of data
* delimiter=';' - data have more ; symbols. to remove them used delimiter.
```

In [57]:

```
# data=pd.read_csv("richies.csv", engine='python', delimiter='"',error_bad_lines=False)
data=pd.read_csv("richies.csv",delimiter=',')
data
```

Out[57]:

	Name	NetWorth	Age	Country	Source	Industry
0	Elon Musk	$219,000,000,000	50	United States	Tesla, SpaceX	Automotive
1	Jeff Bezos	$171,000,000,000	58	United States	Amazon	Technology
2	Bernard Arnault & family	$158,000,000,000	73	France	LVMH	Fashion & Retail
3	Bill Gates	$129,000,000,000	66	United States	Microsoft	Technology
4	Warren Buffett	$118,000,000,000	91	United States	Berkshire Hathaway	Finance & Investments
...	...	...	...	...	...	...
96	Vladimir Potanin	$17,300,000,000	61	Russia	metals	Metals & Mining
97	Harold Hamm & family	$17,200,000,000	76	United States	oil & gas	Energy
98	Sun Piaoyang	$17,100,000,000	63	China	pharmaceuticals	Healthcare
99	Luo Liguo & family	$17,000,000,000	66	China	chemicals	Manufacturing
100	Peter Woo	$17,000,000,000	75	Hong Kong	real estate	Real Estate

101 rows × 6 columns

In [59]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101 entries, 0 to 100
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Name      101 non-null    object
 1   NetWorth  101 non-null    object
 2   Age       101 non-null    int64 
 3   Country   101 non-null    object
 4   Source    101 non-null    object
 5   Industry  101 non-null    object
dtypes: int64(1), object(5)
memory usage: 4.9+ KB
```

In [60]:

```
data.head()
```

Out[60]:

	Name	NetWorth	Age	Country	Source	Industry
0	Elon Musk	$219,000,000,000	50	United States	Tesla, SpaceX	Automotive
1	Jeff Bezos	$171,000,000,000	58	United States	Amazon	Technology
2	Bernard Arnault & family	$158,000,000,000	73	France	LVMH	Fashion & Retail
3	Bill Gates	$129,000,000,000	66	United States	Microsoft	Technology
4	Warren Buffett	$118,000,000,000	91	United States	Berkshire Hathaway	Finance & Investments

In [61]:

```
data.tail()
```

Out[61]:

	Name	NetWorth	Age	Country	Source	Industry
96	Vladimir Potanin	$17,300,000,000	61	Russia	metals	Metals & Mining
97	Harold Hamm & family	$17,200,000,000	76	United States	oil & gas	Energy
98	Sun Piaoyang	$17,100,000,000	63	China	pharmaceuticals	Healthcare
99	Luo Liguo & family	$17,000,000,000	66	China	chemicals	Manufacturing
100	Peter Woo	$17,000,000,000	75	Hong Kong	real estate	Real Estate

In [62]:

```
s=data.drop(["Age"],axis='columns',inplace=True)   #remove or del col
```

In [63]:

```
data.isnull().sum()   # check the null values
```

Out[63]:

```
Name        0
NetWorth    0
Country     0
Source      0
Industry    0
dtype: int64
```

In [ ]:

```
data.dropna(axis=0,inplace=True)  # remove null values. After learn Stats,
#we will not remove nulls we can correct them
# inplace: If it is True, then it replaces in place.
```

In [65]:

```
data['Country'].values
```

Out[65]:

```
array(['United States', 'United States', 'France', 'United States',
       'United States', 'United States', 'United States', 'United States',
       'United States', 'India', 'India', 'United States', 'Mexico',
       'France', 'United States', 'United States', 'China',
       'United States', 'United States', 'Canada', 'United States',
       'United States', 'Spain', 'United States', 'China', 'Canada',
       'United States', 'Germany', 'Hong Kong', 'United States', 'France',
       'France', 'Germany', 'China', 'Germany', 'Italy', 'Hong Kong',
       'United States', 'Hong Kong', 'United States', 'United States',
       'United States', 'France', 'France', 'Mexico', 'Australia',
       'India', 'United States', 'China', 'United States', 'Austria',
       'Italy', 'United States', 'Japan', 'China', 'Germany', 'India',
       'China', 'China', 'United States', 'Japan', 'China', 'France',
       'Indonesia', 'United States', 'Switzerland', 'Chile', 'China',
       'Indonesia', 'United States', 'United States', 'United States',
       'United States', 'Japan', 'United States', 'United States',
       'Germany', 'United States', 'China', 'United States', 'India',
       'China', 'China', 'Germany', 'Germany', 'China', 'Russia', 'China',
       'India', 'Australia', 'China', 'India', 'China', 'Singapore',
       'Sweden', 'United States', 'Russia', 'United States', 'China',
       'China', 'Hong Kong'], dtype=object)
```

In [ ]:

```
data=data.replace('Canda','Canada',regex=True)  # As we are operating on strings, put regex
data.head()
data.dtypes
# regex: For pandas to interpret the replacement as regular expression replacement, set it
```

In [66]:

```
data=data.replace(['\$'," "],"",regex=True)   # Removes $ symbol before money
data
```

Out[66]:

	Name	NetWorth	Country	Source	Industry
0	ElonMusk	219,000,000,000	UnitedStates	Tesla,SpaceX	Automotive
1	JeffBezos	171,000,000,000	UnitedStates	Amazon	Technology
2	BernardArnault&family	158,000,000,000	France	LVMH	Fashion&Retail
3	BillGates	129,000,000,000	UnitedStates	Microsoft	Technology
4	WarrenBuffett	118,000,000,000	UnitedStates	BerkshireHathaway	Finance&Investments
...	...	...	...	...	...
96	VladimirPotanin	17,300,000,000	Russia	metals	Metals&Mining
97	HaroldHamm&family	17,200,000,000	UnitedStates	oil&gas	Energy
98	SunPiaoyang	17,100,000,000	China	pharmaceuticals	Healthcare
99	LuoLiguo&family	17,000,000,000	China	chemicals	Manufacturing
100	PeterWoo	17,000,000,000	HongKong	realestate	RealEstate

101 rows × 5 columns

In [ ]:

```
def value_to_float(x):
    if 'K' in x:
        if len(x) > 1:
            return float(x.replace('K','')) * 0.000001
    if 'M' in x:
        if len(x) > 1:
            return float(x.replace('M','')) * 1000
    if 'B' in x:
        if len(x) > 1:
            return float(x.replace('B','')) * 100000
data['$ Last Change'] = data['$ Last Change'].apply(value_to_float)

# '$ Last Change' is column name. We can put required column here based on the dataset
```

In [67]:

```
data.dtypes
```

Out[67]:

```
Name        object
NetWorth    object
Country     object
Source      object
Industry    object
dtype: object
```

In [ ]:

```
display(data[data['NetWorth']>=50])
```

In [68]:

```
display(data[data.Country=='India'])
```

	Name	NetWorth	Country	Source	Industry
9	MukeshAmbani	90,700,000,000	India	diversified	Diversified
10	GautamAdani&family	90,000,000,000	India	infrastructure,commodities	Diversified
46	ShivNadar	28,700,000,000	India	softwareservices	Technology
56	CyrusPoonawalla	24,300,000,000	India	vaccines	Healthcare
80	RadhakishanDamani	20,000,000,000	India	retail,investments	Fashion&Retail
88	LakshmiMittal	17,900,000,000	India	steel	Metals&Mining
91	SavitriJindal&family	17,700,000,000	India	steel	Metals&Mining

In [69]:

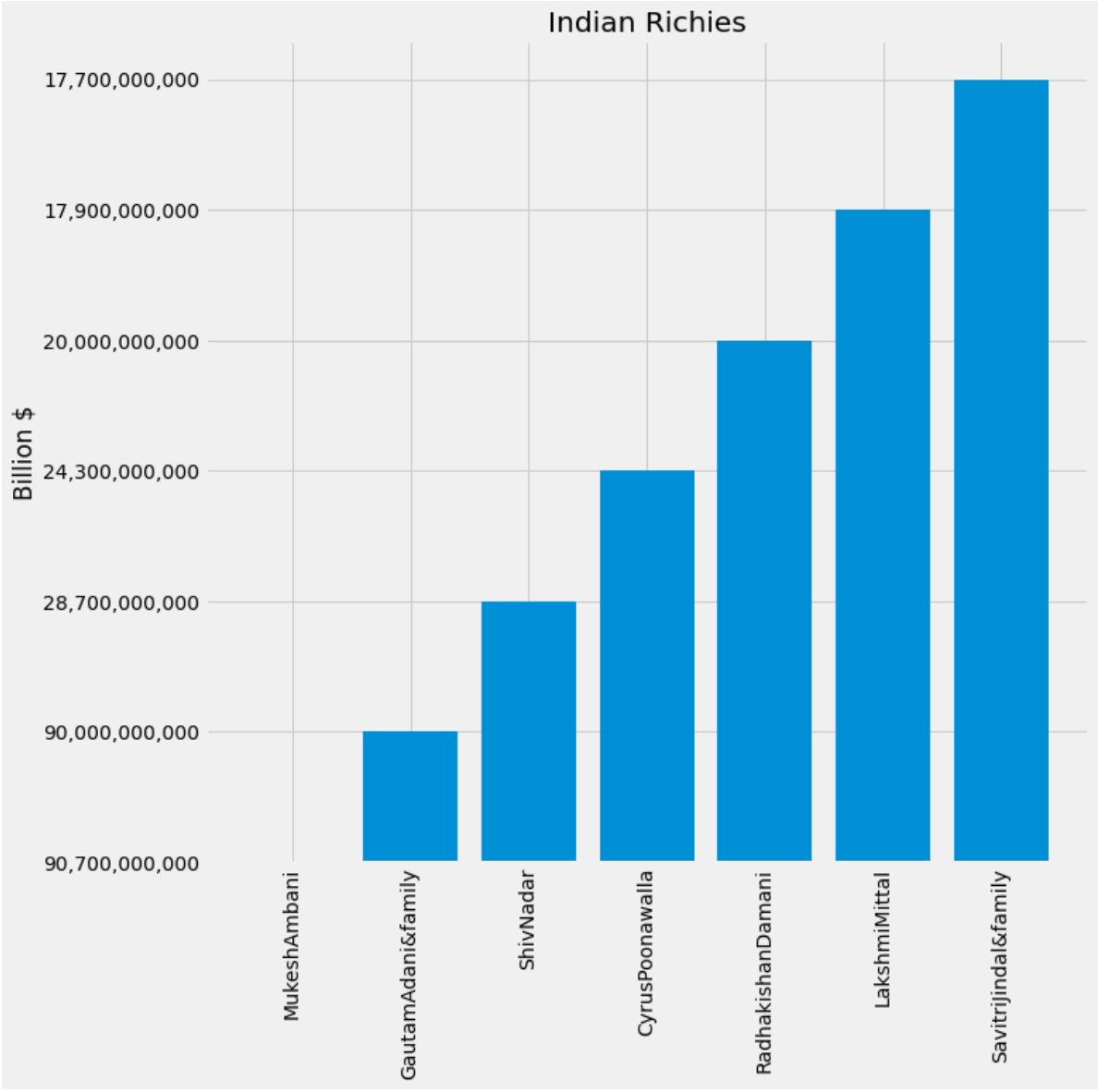
```
import matplotlib.pyplot as plt
from matplotlib import style
from matplotlib import figure
style.use('fivethirtyeight')
```

In [71]:

```
india=data[data.Country=='India']
```

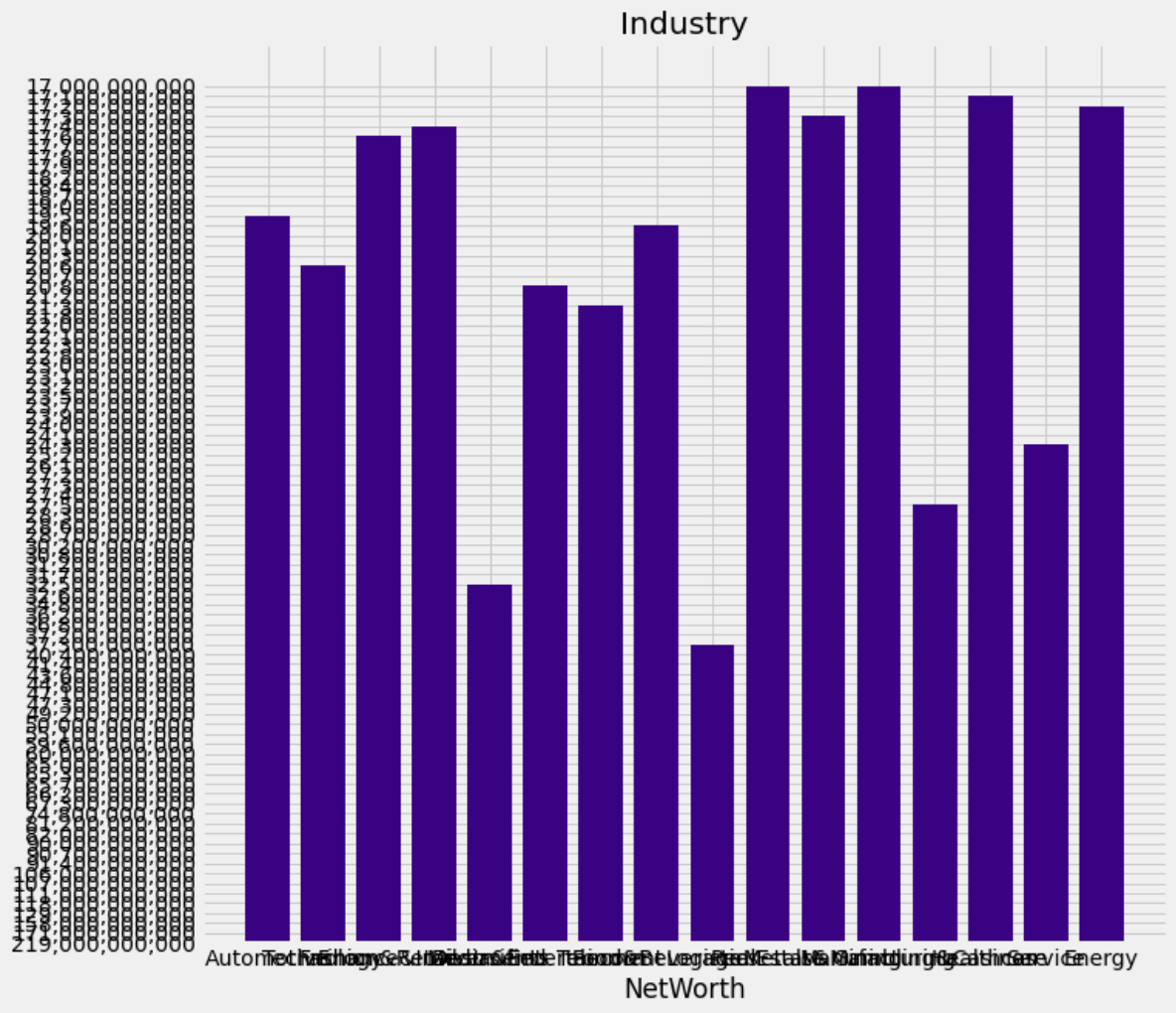
In [77]:

```
plt.bar(india['Name'],india['NetWorth'])    # we gave columns here instead of direct data l
plt.title('Indian Richies')
plt.ylabel('Billion $')
plt.gcf().set_size_inches(10,10)        # gcf(get current figure) - used to increase/decrea
plt.xticks(rotation=90)
plt.show()
```



In [89]:

```
# OF All Countries
plt.bar(data['Industry'],data['NetWorth'],color='#380282')
plt.title("Industry")
plt.xlabel("NetWorth")
plt.gcf().set_size_inches(10,10)
plt.show()
```



In [81]:

```
india['Industry']
```

Out[81]:

```
9        Diversified
10       Diversified
46        Technology
56        Healthcare
80    Fashion&Retail
88     Metals&Mining
91     Metals&Mining
Name: Industry, dtype: object
```

In [82]:

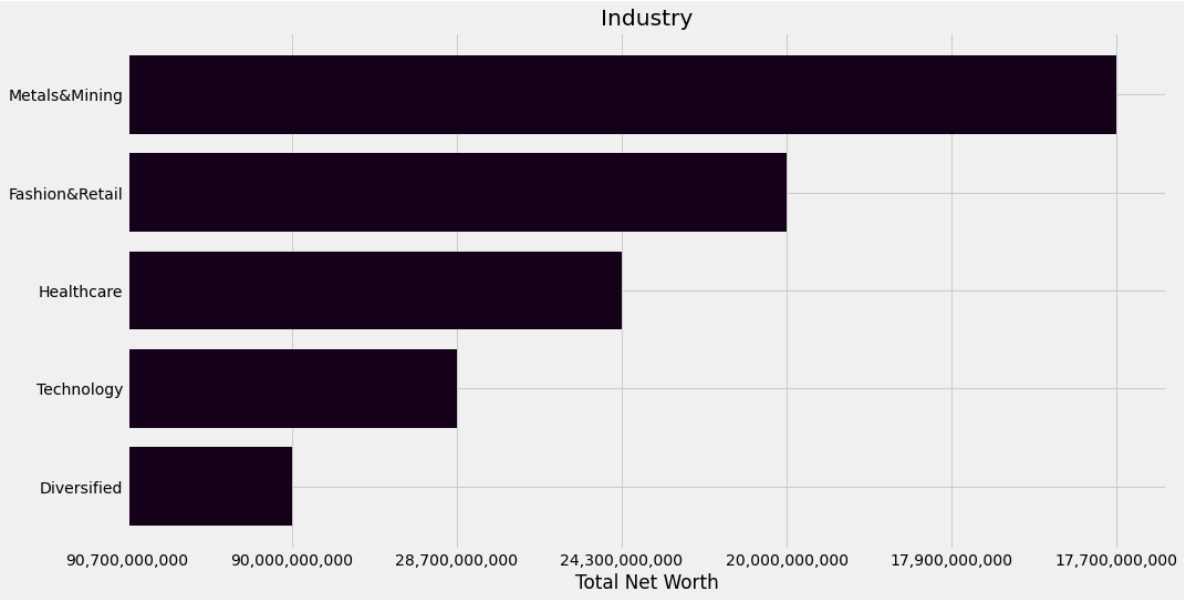
```
india
```

Out[82]:

	Name	NetWorth	Country	Source	Industry
9	MukeshAmbani	90,700,000,000	India	diversified	Diversified
10	GautamAdani&family	90,000,000,000	India	infrastructure,commodities	Diversified
46	ShivNadar	28,700,000,000	India	softwareservices	Technology
56	CyrusPoonawalla	24,300,000,000	India	vaccines	Healthcare
80	RadhakishanDamani	20,000,000,000	India	retail,investments	Fashion&Retail
88	LakshmiMittal	17,900,000,000	India	steel	Metals&Mining
91	SavitriJindal&family	17,700,000,000	India	steel	Metals&Mining

In [83]:

```
# India only
plt.barh(india['Industry'],india['NetWorth'],color='#15001A')
plt.xlabel("Total Net Worth")
plt.title("Industry")
plt.gcf().set_size_inches(15,8)
plt.show()
```



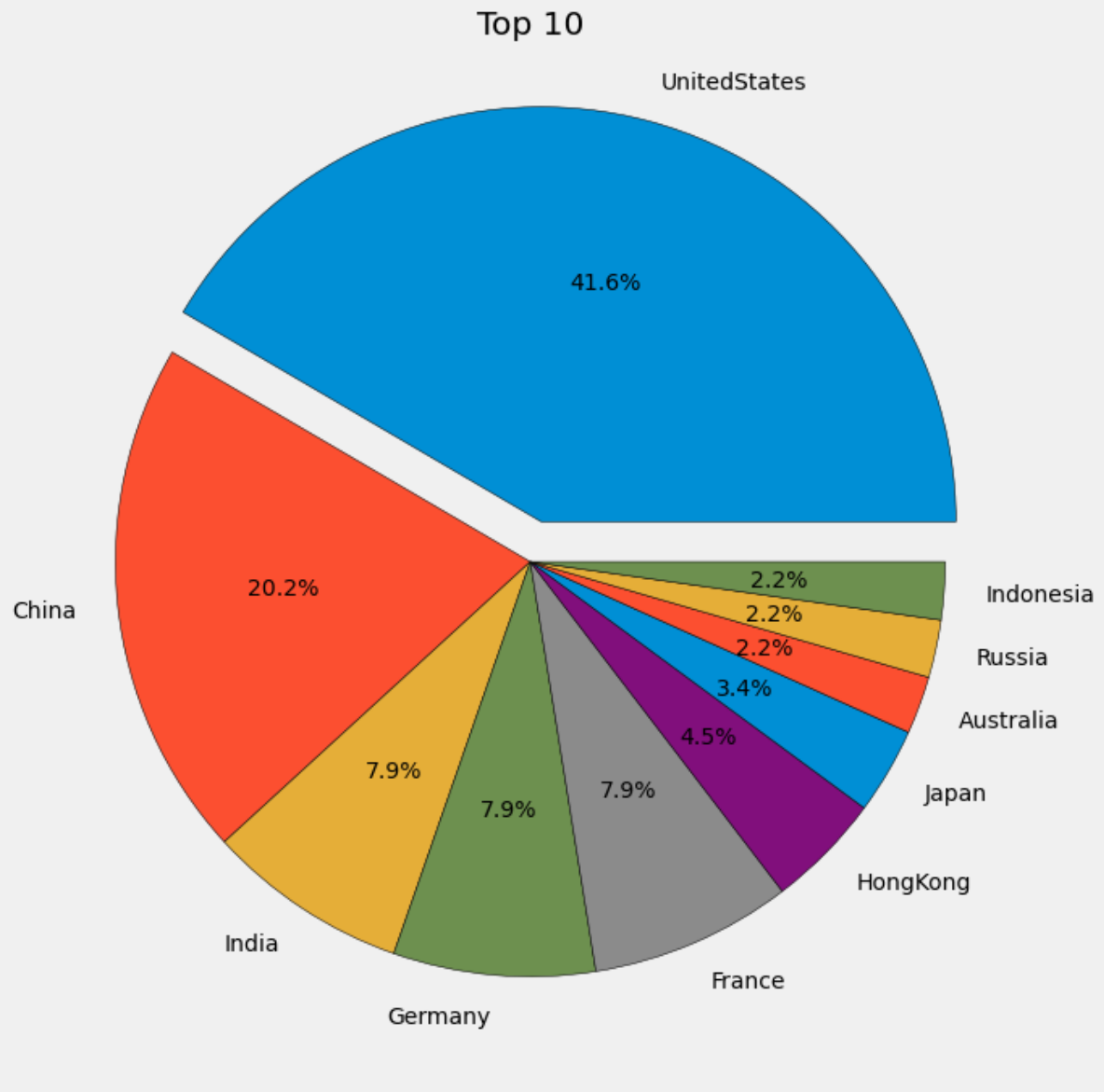
In [84]:

```
display(data["Country"].value_counts())
```

```
UnitedStates    37
China           18
India            7
Germany          7
France           7
HongKong         4
Japan            3
Australia        2
Russia           2
Indonesia        2
Italy            2
Canada           2
Mexico           2
Austria          1
Spain            1
Switzerland      1
Chile            1
Singapore        1
Sweden           1
Name: Country, dtype: int64
```

In [85]:

```
# Percentage of richies from top 10 contries
country=data['Country'].value_counts().head(10).values
name=data["Country"].value_counts().head(10).index
plt.gcf().set_size_inches(20,11)
plt.pie(country,labels=name,autopct="%1.1f%%",wedgeprops={'edgecolor':'black'},explode=[0.1
plt.title('Top 10')
plt.show()
```



In [86]:

```
data[data.Country=='UnitedStates'].head(10)
```

Out[86]:

	Name	NetWorth	Country	Source	Industry
0	ElonMusk	219,000,000,000	UnitedStates	Tesla,SpaceX	Automotive
1	JeffBezos	171,000,000,000	UnitedStates	Amazon	Technology
3	BillGates	129,000,000,000	UnitedStates	Microsoft	Technology
4	WarrenBuffett	118,000,000,000	UnitedStates	BerkshireHathaway	Finance&Investments
5	LarryPage	111,000,000,000	UnitedStates	Google	Technology
6	SergeyBrin	107,000,000,000	UnitedStates	Google	Technology
7	LarryEllison	106,000,000,000	UnitedStates	software	Technology
8	SteveBallmer	91,400,000,000	UnitedStates	Microsoft	Technology
11	MichaelBloomberg	82,000,000,000	UnitedStates	BloombergLP	Media&Entertainment
14	MarkZuckerberg	67,300,000,000	UnitedStates	Facebook	Technology

In [ ]:

```
#Last Change in Inda
plt.bar(india['Name'],india['$ Last Change'],color=(india['$ Last Change']>0.0).map({True:'
plt.xlabel("Indian Richies")
plt.ylabel("$ Last Change")
plt.gcf().set_size_inches(15,8)
plt.xticks(rotation=90)
plt.show()
```

In [ ]:

```
* All this is EDA without Stats.
```

In [ ]:

```
Task 43:
    Practice matplotlib

Task 44:
    Take 3 datasets and perform EDA process

Task 45:
    Explore html
```

Python - Threads & regex

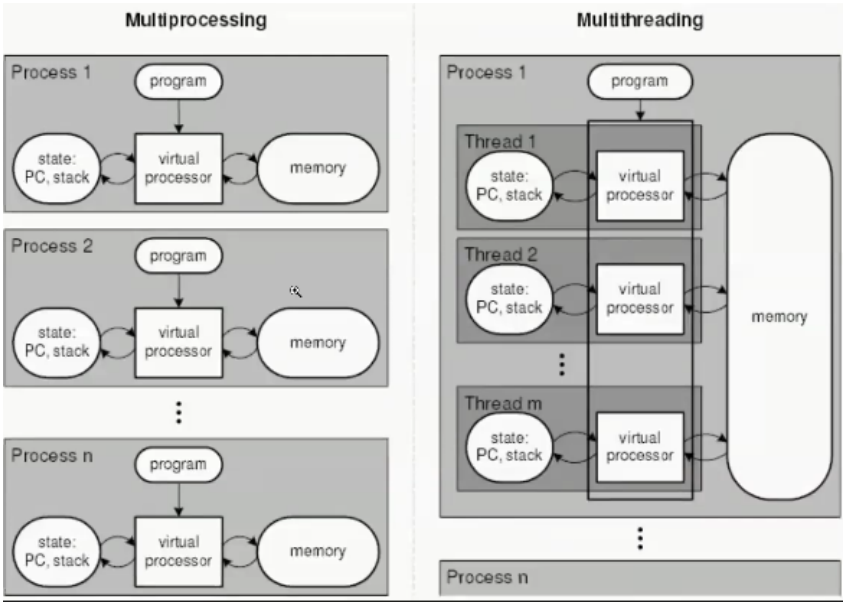
Aug 23. Tue

Process:

Process is a program(set of instructions) in execution. Process cannot share the memory.

Thread:

Thread is a light-weight process. Threads can be used to perform complicated task in the background without interrupting the main program. Threads can share the memory.



multiprocessing vs multithreading:

* multiprocessing can have multiple processes and each contains its individual memory. They won't share memory between processes.

* Single process can have multiple threads and all these processes will share single memory.

* Both the multiprocessing and multithreading will process in OS.

* Thread life cycle is same as human life cycle.

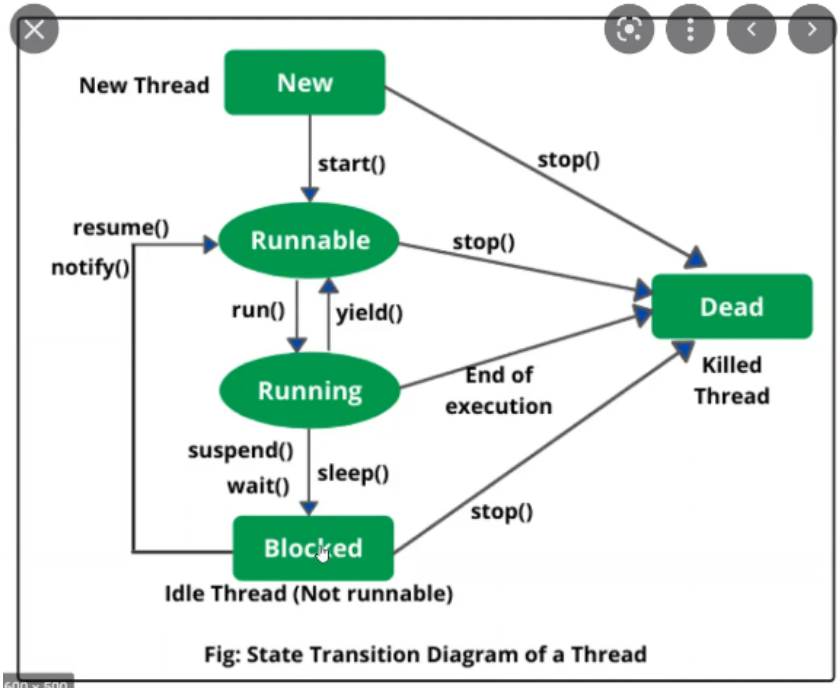


Fig: State Transition Diagram of a Thread

run(): It is the entry point for a thread.
start(): method starts a thread by calling the run method.
join(time in milliseconds): Waits for threads to terminate.
is_alive(): checks whether thread is still executing or not. gives True/Fasle.
getName(): method returns the name of athread.

```
import threading
from threading import *
from time import sleep

class Ravi(Thread):
        def run(self):
                for i in range(10):
                        print(i)
```

import threading - threading is a predefined class
Ravi(Thread) - When we pass Thread(derived class from threading) in class. normal class will changed to Thread class.

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

```
import threading
from threading import *
from time import sleep

class Ravi(Thread):
    def run(self):      # run() is predefined function here
        for i in range(10):
            print('Ravi thread 1')
            sleep(2)

class Ajay(Thread):
    def run(self):
        for i in range(10):
            print('Ajay thread 2')
            sleep(2)
n=Ravi()
#print(n.is_alive())   # it will give False here because thread not started yet.
# n.run()  # Generally. we will call like this
nl=Ajay()
# nl.run()
n.start()
# print(n.is_alive())    # it will give True now becuase thread already started.
sleep(1) # If we start() thread. we can observer how thread running
nl.start()
n.join()  # terminate
print(n.is_alive())     # when we use join() thread terminated. Hence is_alive()
showing False.

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

# If we consider ticket booking. system should give same priority one after
other alternatively.
# This is not happening here now. Threads are running randomly.
# we will use sleep() in both class and function call to achieve same priority
to 2 persons
#-------------------------------
```

If no threads in out program. it will run in default main thread.

print(threading.current_thread().getName()) - used to check the thread name.

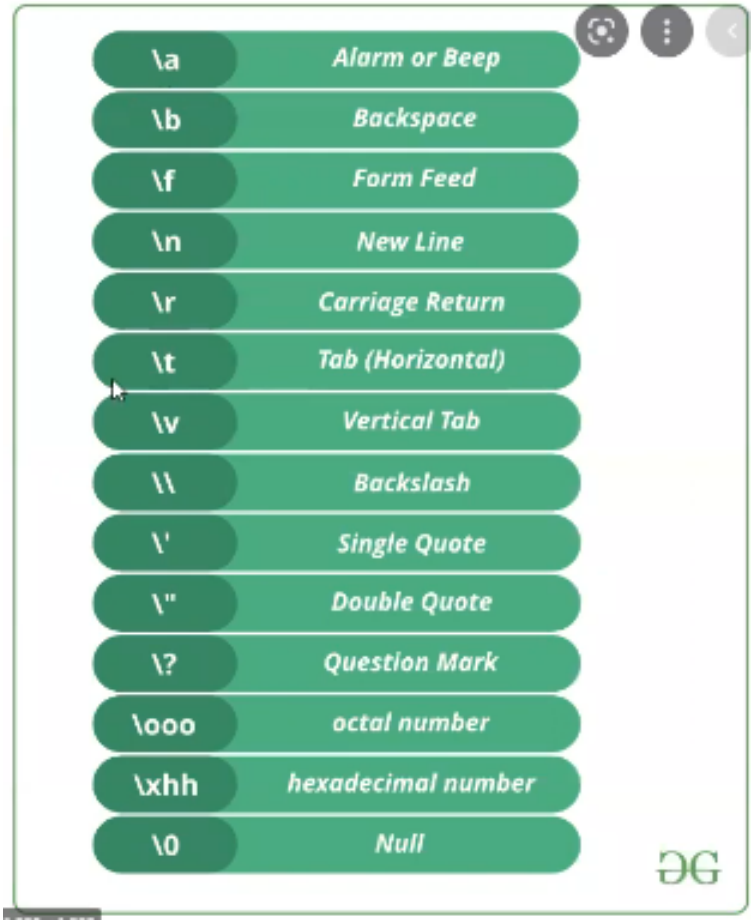
Regex:

The Regex or Regular Expression is a way to define a pattern for searching or manipulating strings. We can use a regular expression to match. search. replace. and manipulate inside textual data.

This is used while taking our information in website forms. This patterns will check whether given no is ten digits or not like that.

Raw String:

escape sequences(\n) will not work in raw strings. It will print them as it is. if we take escape sequences in normal strings. it will give new line.



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

Eg: print('kiran')
print(r'kiran') # we can use r or R
print(r'kiran'=='kiran')

Results True as both are same

print(r'the \n the')

Output: the \n the

```
print('the \n the')
```

Output: the
the

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

Find all:

re.findall() method scans the regex pattern through the entire target string and returns all the matches that were found in the form of a list.

Regular- Expression Patterns

^	Matches beginning of line.
$	Matches end of line.
.	Matches any single char except newline.
[...]	Matches any single char in brackets.
[^...]	Matches any single char not in brackets.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace.
\S	Matches nonwhitespace.
\d	Matches digits.
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string.
\z	Matches end of string.
\G	Matches point where last match finished.
x| y	Matches either x or y.

[0-9]	Match any digit; same as [0123456789]
[a-z]	Match any lowercase ASCII letter
[A-Z]	Match any uppercase ASCII letter
[a-zA-Z0-9]	Match any of the above
[^aeiou]	Match any other than a lowercase vowel
[^0-9]	Match anything other than a digit.

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

```
import re
ip_s=re.findall(r'\d{1.3}.\d{1.3}.\d{1.3}.\d{1.3}'.s)
print(f'ip address are -: {ip_s}')
```

Output:
ip address are -: ['172.45.78.109'. '127.0.0.1'. '10.67.89.101'. '11.67.98.102'. '12.68.98.102']

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

Here.
import re - regular expresssion
(r'\d{1.3}.\d{1.3}.\d{1.3}.\d{1.3}'.s) - r for raw string.
\d - search all digits
{1.3} - either 1/2/3 digits
s - before expression will work on this s string

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

matches=re.findall(r"[A-Z][a-z]+\s\d{1.2}","Those are the match dates June 24. August 9. Dec 12")
print(f'gives Month Date format - {matches}')

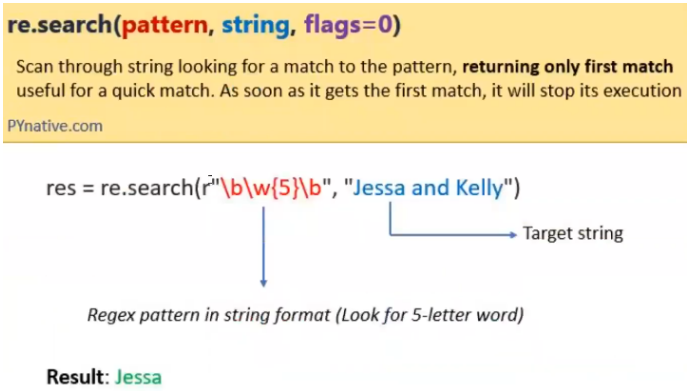
##################################

Here.
(r"[A-Z][a-z]+\s\d{1.2}" -
[A-Z] - first system checks for Capital letter words in given raw string "Those are the match dates June 24. August 9. Dec 12"
- It will get T. J. A. D
[a-z] - next letters should be small letters.
- It will get Those. June. August. Dec
\s - after wods it should be space
- It will get Those. June. August. Dec
\d{1.2}- after space there should be either one or two digit nos.
- It will get June 24. August 9. Dec 12

(r"[A-Z][a-z]+\s(\d{1.2})" - If we want only digts put those digit pattern in ().
(r"([A-Z][A-Z]+\s(\d{1.2})" - If we want data tuple format like below.
- [('June'.'24').('August'.'9').('Dec'.'12')]

search:

The re.search() returns only the first match to the pattern from the target string.



```
##################################

        import re
        target_string="Emma is a python developer \n Emma also knows ML and AI"
        # caret (^) matches at the beginning of a string
        result=re.search(r"^\w{4}".target_string)
        print(result)
        print(result.group())

        Output:
        <re.Match object: span=(0, 4), match='Emma'>
        Emma
##################################

        Here,
        ^ - beginning of the string
        \w{4} - Word with 4 letters i.e. Emma.
        Although we have another Emma, search method will return only first value
        only.

##################################
        import re
        strl="Emma is a python developer \n Emma also knows ML and AI"
        # dollar sign($) to match at the end of the string
        result=re.search(r"\w{2}$".strl)
        print(result.group())

        Output:
        AI

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

        Here,
        $ - ending of the string
        \w{2}$ - two digit word from at the end of string
        Although we have MI with same criteria, it will give first occurance from
        last.

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

        import re
        st='In a world where you can be anything, be king'
        match_object=re.search('In'.st)
        print(f'type is object {match_object}')

##################################
```

Here.
In - We can find 'In' in the given string. it will give the position of 'In' also.

match_object.start() - 0
match_objectl.start() - 25
match_object.span() - (0.2)
match_objectl.span() - (25.27)

sub(substitute):

It is like same as replace() function.

sub('old pattern','new pattern',source_str)

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

```
import re
st='In a world where you can be anything. be king'
sb=re.sub('e','E',st)
print(sb)
```

Output:
In a world whErE you can bE anything. bE king

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

```
import re
st='In a world where you can be anything. be king'
sbl=re.sub('e','E',st,3)
print(sbl)
```

this will replace 'e' to 'E' 3 times only.

Complie:

The re.compile() method changed the string pattern into a re.Pattern object that we can work upon.

```
import re
a='hat mat rat pat'
reg=re.complie('[r]at')
print(reg)
```

If we write a pattern and store to pattern object using compile. we can use it whenever you need without writing pattern again.

working with white spaces:

\b - backspace
\f - formfeed
\r - carriage return
\t - tab
\v - vertical

```
import re
comp = re.compile('\n')
new=comp.sub('.chelsea)
print(new)
```

Match:

re.match() method looks for the regex pattern only at the beginning of the target string and returns match object if match found: otherwise. it will return None.

match() vs search() vs findall():

match() - will match 0 index. if pattern not found. it will return null
search() - will match all the words in string but show only first occurance of pattern
findall() - match all the words of string and returns whatever words matched with pattern

****

Task 46 :

practice threading and regex

Task 47 :

Complete 6 problem solving youtube videos

Task 48 :

explore html (anchor. img. videos)

Python - Power BI

Aug 24. Wed

- adibogadi

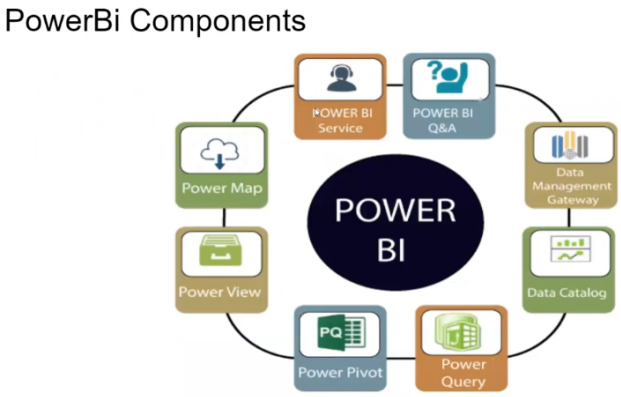
BI:

Business Intelligence(BI) refers to the applications. technologies and practices. For the collection. analysis. integration and presents the business information. The purpose of business intelligence is to support better decision making.

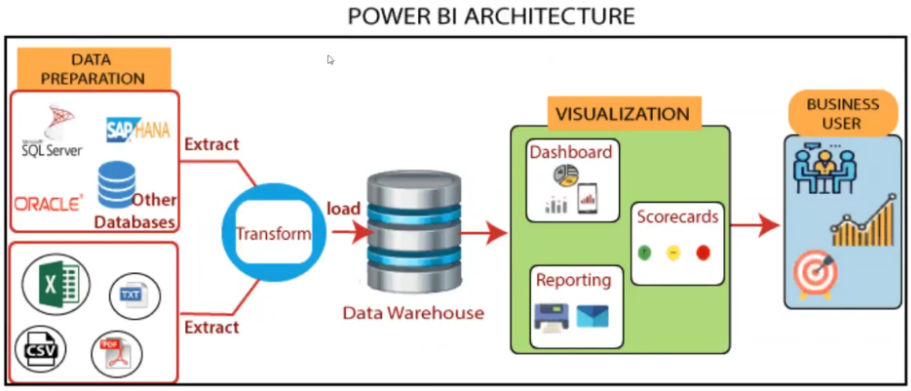
Power BI:

Power BI is a Data Visualization and Business Intelligence tool which helps to convert data from different data sources into interactive dashboards and BI reports. It provides interactive visualizations with self-service business intelligence capabilities where end users can create reports and dashboards by themselves. without having to depend on information technology staff or database administrators.

Power BI Components:



Power BI Architecture:



We will perform Extraction. Transformation. Loading (ETL) process using Power BI.

****

Task 49 :

Take 3 datasets and perform DV using Power BI

Task 50 :

Prepare HTML

Task 51 :

Prepare a resume

We have to mention like this in resume:

Skills:
Programming Languages: Java. C Programming. Basic to Advanced Python.

Python Libraries: Numpy. Pandas. Matplotlib

Having knowledge on OOPs in Python
Having knowledge on EDA
Having knowledge on Multithreading. regex and webscrapping.
Tools: Power BI

Projects: Mention minimum 3 projects.

Project Name:
Description:
Responsibilities:
Tools Used:
Github Link:

Practice problem solving videos in Python life channel