UNIT - 1

1.1 Introduction To Python :.

In the python Introduction that includes that all about the python programming.

What is python :

1. python is a popular programming Language.
2. Created by Guido - Van - Rassum
3. Introduced in the year 1991. ✓ 1981
4. It's mainly useful for the...
   a. Web Development [Server-Side]
   b. Software Development
   c. Mathametics.
   d. System Scripting.

What Can python do.

1. python can be used on a Server to creates a web application.
2. python can be used alongside Software to create workflows.
3. python can Connect the database Systems.
4. It Can also read & modify the files.
5. It Can used to handle big data & perform Complex mathematics.
6. python Can be used for "Rapid prototyping", or for production-ready Software development.

Why Python :

1. python works on different platforms i.e windows, MAC, Linux, PI, Raspberry and etc...
2. python has a simple syntax similar to the English language.
3. python has syntax that allows developers to write programs with fewer lines than some other programming language.
4. Python runs on a Interpeter System, meaning that code can be Executed as soon as it is written This means that prototyping can be very quick.
5. Python can be treated in a procedural way, an object oriented way or a functional way.

Python Trending Now...

✓ The most recent major version of python is "Python 3", which we can shall be using in the "ANACONDA NAVIGATOR". However, "python 2" although not being updated with anything other than security updates, is still quit popular.

✓ It is possible to write python in an a Integrated Development Environment [IDE], such as Thonny, pycharm, Netbeans or Eclipse which are particularly useful when managing large collections of the Python file.

Python Syntax Compared with other programming

1. python was designed for readability.
2. It have some similarities like English language with influence from mathematics.
3. It uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parantheses.
4. Python relies on "Indentation" using whitespace to define scope; such as the loops, functions & classes. other programming languages often use "Curly-brackets" for this purpose.

Sample Code:-

```
print("JNTUK-UCEV")
```

o/p: JNTUK-UCEV

Python Get Started:

Python Install:

1. Many PC, MACs will have python already installed

```
c:\Users\Your Name>python --version.
```

2. To check that python is installed on the Linux/MAC then the linux open the command line or Mac open that the Terminal & type:

```
python --version.
```

Python Quick Start :

Python is a Interpreted programming language. The developer can write and save that the file with the Extension: **.py** in the text Editor. and then put those files into the python interpreter to be executed.

The way of run the python file.

```
c:\users\chandu\ mypythonprog.py
```

where :

mypythonprog.py - filename of python

The Python Command Line :-

To test a short amount of code in python. Sometimes it is quickest and easist, now to write the code in the file.

"This is made possible because python can be run as command line itself."

To get the python Terminator proceed :/

```
c:\users\chandu\>python.
```

— If its not working

```
c:\users\chandu > py
```

Then you can see that the Editor with ">>>" and if you want to close that terminator write the **exit()**

Terminator to run the code :

```
c:\users\chandu> python
python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017,06:04:4
[MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license"
for more information.
>>> print("JNTUKV")
JNTUKV
```

Python Indentation :

1. Indentation refers to the spaces at the begining of a code line.

2. Where in other programming language the indentation in code for readability only that the indentation in python is very important.

3. Python uses indentation to indicate a block of a code.

Ex : 1

```
if a>b:
    print("a is big")
```

o/p : a is big

Ex : 2

```
if a>b:
print("a is big")
```

o/p : Error, because no indentation.

1.2 Python Program Development Cycle:

Introduction:

1. Program development life cycle is a "outline" of the each steps used to build a software application.

2. The program development life cycle is a tool used to Guide Computer programmers through the development of an Application.

There are 6 steps for "PDLC" in Python.

1. Analyze the problem
2. Design the program
3. Code the program.
4. Test & Debug the program
5. Formalize the Solution.
6. Maintain the Program.

1. Analyze the Problem:

Define the problem to be solved, and write program Specifications, descriptions of that the program inputs, processing, out and user interface.

2. Design the Program:

Draw the flow Chart with the program Specifactions and the description, Come up with programming code.

3. Code the program:

We need to write the code / develop the code with based on flowcharts, program Specifications & program descriptions.

4. Test and Debug the Code:

Testing:

It is a process to find the bugs & Errors

Debug:

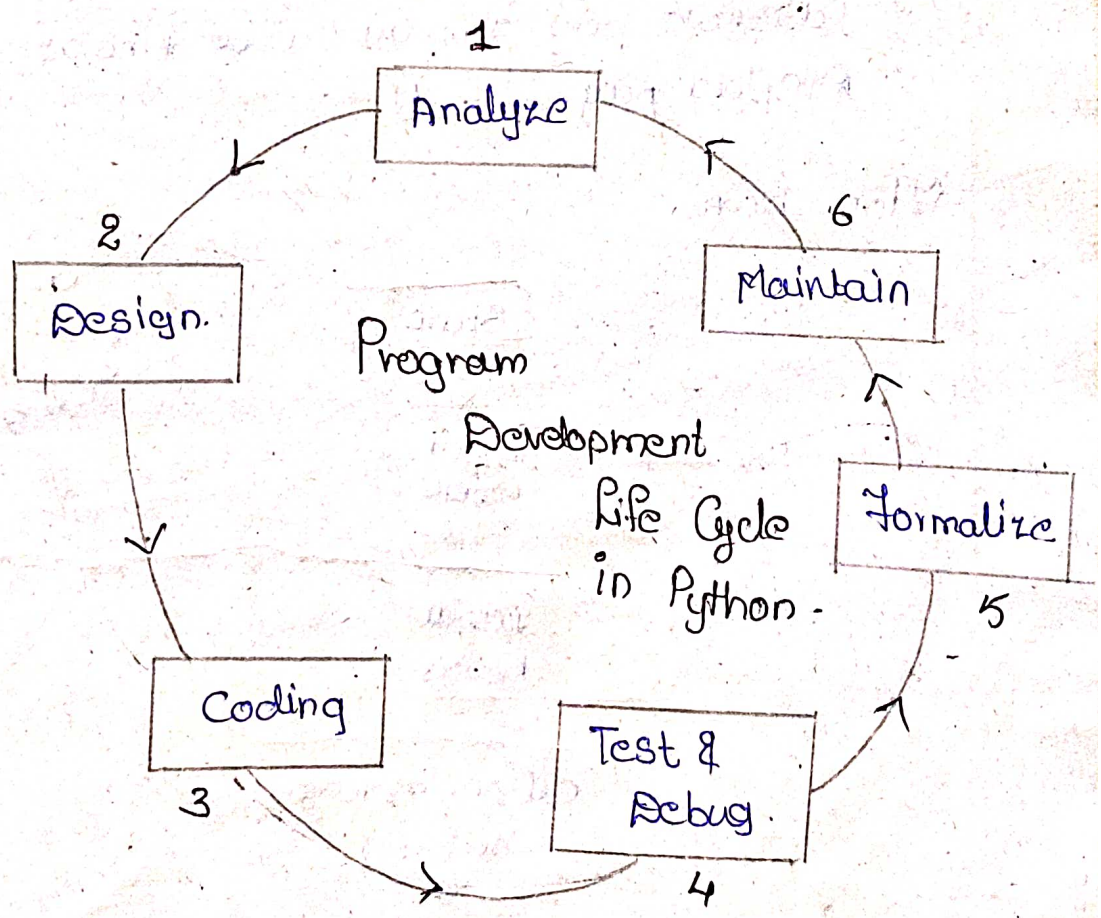
It is the process of fixing the Errors and make that Errors should be zero.

5. Formalize the Solutions:

we have to maintain that the program to the "Error free".

6. Maintain the program:

Once your application is used in the realtime and make that application to not get the Error when the End user..



1.3: Python Input, Processing, Output : [IPO]

In this topic we covers that the "PDLC" design-aids(pseudocode and flowcharts).

Input, processing, output [IPO]

Variables, Datatypes, literals and etc...

Sample Program Development Life Cycle.

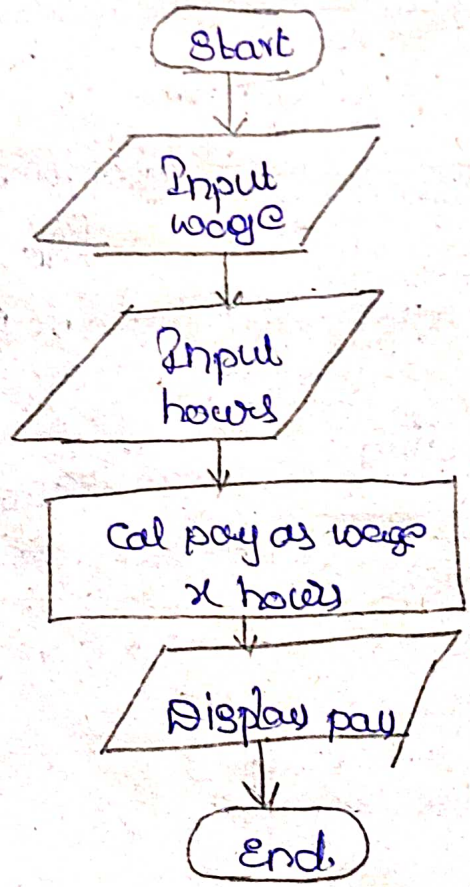
Overview :

Analyze → Design → Code → Test/Debug → Document

Pseudocode :

- ✓ Get wage from user
- ✓ Get hours worked from user
- ✓ Calculate pay amount : wage * hours
- ✓ Display pay amount.

Flow Chart :



Input, Processing, Output :

Get data from the using : input.

Display data to the using : print.

Both of these need to be followed by parenthesis

use apostrophes or quotes around literal. text (aka "strings")

print(Hello) - wrong

print("Hello") - Correct.

Result of that the user input needs to be stored somewhere and that be put on the left side.

Ex:

username = input("Enter your name :")

- In this way we read the data from user.

1.4: Comments in Python :

1. Comments can be used to explain the code.
2. Comments can be used to make the code to more Readable.
3. Comments can be used to prevent the execution when testing the code.

Types of Comments:

a. Single line
b. Multi line

a. Single line Comments :

Its denoted with Symbol Hash "#"

Ex: #This is my first program.
print(" VCEV")

b. Multiline Comments :

Python does not really have a Syntax for the multi line Comments.

To add a [#] to multiple lines, so, its a long process instead of that we can use the "three double Quotes Symbol Continously """"

Ex : """
This is first program
Iam ok
To with this code. """
print("Hello Vizinagaram").

Note :

As long as the String is not assigned to a variable, python will be read the code, but then ignore it, and you have made Multi line Comments.

1.5 Python Variables :

Variable :

A Variable is a name that can Stores a value that defined type.

Variables are the Containers for Storing the data values.

Creating a Variables :

python has no Command for declaring a Variable.

A variable is Created the moment you first assign a value to it.

Example-1:

We can assign any type values to that variable

```
x = 5
y = 10.5
z = "python"
print(x, y, z)
```

O/p : 5 10.5 python.

Casting:

If you want to specify the data type of a variable, this can be done with casting.

Ex:
```
x = str(3)
y = int(3)
z = float(3)
print(x)
print(y)
print(z)
```

output: 3
3
3.0

Get the type:

You can get the datatype of variable with the type() function.

Ex:
```
a = 5
b = "chandu"
print(type(a))
print(type(b))
```

O/p:
```
<class 'int'>
<class 'str'>
```

By Single or Double Quotes:

String variables can be declared either by using the single quote or Double quote.

Ex: a = "Ram" ✓
b = 'Ram' ✓

o/p :- Ram

Case Sensitive:

Variable names are the Case-sensitives.

Ex: a = 4
A = "Hai"

print(a)
print(A)

Both are not Same.

o/p: 4
Hai

Rules for Variable Names:

1. A var name must starts with a letter or the underscore character.
2. A var cannot starts with a number
3. A var name can only contains alpha-numaric characters and underscores (A-Z, 0-9, _)
4. A var names are case-sensitive (for Example: "age, Age, AGE are (3) different Variables".

Illegal Names: (X)

2Myname
My-Name
My Name.

Multi words Variable Names :

Variable names with more than one word can be difficult to be read.

there are several techniques you can use to make them more readble.

1. Camel Case :

Each word, expect the first, starts with a capital letter.

sasiR Sasi_Ra

Ex : myVariableName = "Joy"

hailelloword

hailerowbold

2. Pascal Case :

Each word starts with capital letter.

Ex : MyVariableName = "Raj"

3. Snake Case :

Each word be seperated by underscore.

Ex : my_variable_name = "Ram"

Assign the Mutiple Values :

1. many values to many variables :

python allows you to assign values to the multiple variables in on line.

```
Ex :  a, b, c = "Ram", "Raj", "Joh"
      print(a)
      print(b)
      print(c)
```

o/p : Ram
Raj
Joh.

One Value to Multiple Variables :

```
a = b = c = "chandu"
print(a)
print(b)
print(c)
```

O/p :

```
chandu
chandu
chandu
```

Unpack Collections :

If you have a collection of values in a list, tuple etc. Python allows you extract that the values into variables. This is called unpacking.

Ex :

```
subj = ["DS", "DMS", "Py"]
S1, S2, S3 = subj
print(S1)
print(S2)
print(S3)
```

O/p :

```
DS
DMS
Py
```

Output Variables :

The python print statment is often used to output variables. To combain both text and a variable, python uses the '+' character.

Ex :

```
x = "Super."
print("python is" + x)
```

O/p :

```
python is Super.
```

1.6 Reading Data from the user :

we can read the data from the user by the input()

Syntax :

```
var-name = input("prompt")
```

Ex : Sum of 2-Nums :

```
x = input("Enter a")
y = input("Enter b")
# Sum(x+y)
Sum = x+y
print(Sum)
```

o/p : a [2]
b [5]
7

1.7 Python Data Types :

M

Built-in-Datatypes :

1. In programming, data type is a important Concep
2. Variables can store data for different types can do different things.
3. Python has the following data types built-in. by defaults, in these categories.

Text Type	:	Str
Numaric Types	:	int, float, Complex
Sequence Types	:	list, tuple, range
Mapping Type	:	dict.
Set Types	:	set, frozenset
Boolean Type	:	bool
Binary Type	:	byte, bytearray, memoryview.

* Get the data type of var by $\boxed{\text{type()}}$

Examples :

1. Text Type :

a. Str :

```
x = "Hello Guru"
print(x)
print(type(x))
```

2. Numaric Type :

```
a = 10
b = 5.25
c = 9j
print(a)
print(b)
print(c)
```

3. Mapping Type :

✓ dict

```
a = {"Name" : "Ram",
     "age" : 19}
print(a)
```

4. Boolean Type :

```
x = True
print(x)
y = False
print(y)
```

print(a["Name"])

5. Sequence Type :

List :

```
a = [1, 2, 3, 4, 5]
print(a)
```

Tuple :

```
a = (1, 2, 3)
print(a)
```

Range :

```
x = range(6)
print(x)
```

o/p : range(0, 6)

6. Set Type :

Set :

```
a = {"aa", "bb", "cc"}
print(a)
```

o/p : {'aa', 'bb', 'cc'}

Frozenset :

```
a = frozenset({"aa", "bb", "cc"})
print(a)
```

7. Binary Type :

bytes :

```
x = b"Hello"
print(x)
```

o/p : b'Hello'

bytearray :

```
x = bytearray(5)
print(x)
```

o/p : bytearray(b'\x00\x00\x00\x00\x00')

memoryview :

```
a = memoryview(bytes(5))
print(a)
```

o/p : <memory at 0x00D5BF

Python Numbers :

There are 3 python Numbers that are..

1. int
2. float
3. Complex.

1. int :

INT, Int or integer, is a whole number, positive or negitive, without decimals, of the unlimited length.

Code :

```
x = 1
y = 102030405060708090100120130
z = -3252
```

o/p : as it is.

2. float :

Float, or "floating point number" is a num, +ve or -ve. Containing one or more decimals.

Code :

```
x = 35e5          -  3500000.0
y = 12E4          -  120000.0
z = -87.78e100    -  -8.778e+101
a = 10.5          -  10.5
b = 1.01          -  1.01
c = -35.49        -  -35.49
```

3. Complex Number :

Complex num are written with the 'j' as img part.

Code :

```
x = 3+5j   -  (3+5j)
y = 5j     -  5j
z = -5j    -  (-0-5j)
```

1.8. Type Conversions;

we can convert from one type to another type with the int(), float(), complex() methods.

Convert int → float

x = float(10) // 10.0

Convert float → int

x = int(2.8) // 2

Convert int → complex.

x = complex(100) // (100+0j).

Random Number:

Python does not have a random() function to make a random number, but python has a built-in module called random that can be used to make a randome number.

Code:

```
import random
print(random.randrange(1,10))
```

o/p: For Each Execution it will display the random values.

Python Casting:

a = int(10) - 10
b = int(2.0) - 2
c = int("3") - 3

a = float(10) - 10.0
b = float(2.0) - 2.0
c = float("3") - 3.0

a = str(10) - '10'
b = str(2.0) - '2'
c = str("3") - '3.0'

1.9. Python Strings :-

Strings in python are surrounded by either Single quotation marks, or double quotation marks. 'hello' Same as "hello".

We can use the "print( )" to print.

Assign a String to a Variable :

We can assign a Value through a " = " Symbol

Code :

```
a = "Hello"
print(a)
```

Multiline String :

You can assign a multiline string to a Variable by using three quotes.

Code :

```
a = """ Hai
        Hello
        How are you ? """
print(a)
```

or we can use (''') also.

Strings are Arrays :

access through a index value :

Code :

```
a = "Hello world"
print(a[1])
```

o/p : e

Loop through String :

```
for x in "banana":
    print(x)
```

o/p :

```
b
a
n
a
n
a
```

String Length :

```
a = "Hai"
print(len(a))
```

o/p : 3

Check String :

Check "free" is present in the following text.

```
a = "Hai Are you free now"
print("free" in a)
```

o/p : True

String Slicing :- (Start, stop, step)

You can return a range of characters by using the slice syntax.

Specify the start index and the End index, seperated by a colon, to return a part of the string.

Code-1 :

```
a = "Hello world"
print(a[2:5])
```

Consider. Eliminate

HELLO world.
0 1 2 3 4 5 6 7 8 9 10

o/p : llo

Code-2 :

```
a = "Hello world"
    0 1 2 3 4 5 6 7 8 9 10
print(a[:4])
```

Eliminate.

o/p : Hell.

Code-3 :

```
a = "Hello world"
print(a[2:])
```

Last Element Considered.

o/p : llo world.

Code-4 :

```
a = "Helloworld"
    0 1 2 3 4
    0 1 2 3 4
print(a[-5:-2])
```

0 1 2 3 4 5 6 7 8 9
HELLOworld

o/p : wor

String methods in python :

upper( )

```
a = "hello"
print(a.upper())
```

lower( )

```
a = "HELLO"
print(a.lower())
```

Strip( )

- method is used to remove the white Space

```
a = "_Hello world_"
print(a.strip())
```

(_ — white spaces)

replace( )

```
a = "Hello world"
print(a.replace("w","x"))
```

split( )

```
a = " Hai Hello"
print(a.split(" "))
```

o/p : ['Hai', 'Hello']

String Concatenation :

```
a = "Hai"
b = "Hello"
print(a+b)
print(a+" "+b)
```

o/p : Hai Hello

Formate Strings :

We Cant Concatenate the int to Str. but we Can be do by the $\boxed{\text{Formate( )}}$

Code :

```
age = 36
name = " Ram Ram {} "
print(name.format(age))
```

Escape Charactors:

\' - Single Quote
\\ - Backslash
\n - newline
\r - Carriage return
\t - tab
\b - backspace
\f - Form feed
\ooo - octal value
\xhh - Hex Value.

1.10. Operators:

An operator is a symbol that is used to perform particular task.

1. Arithmetic operators
2. Assignment Operators
3. Comparision Operators
4. Logical operators.
5. Identity Operators.
6. Membership Operators
7. Bitwise operators.

1. Arithmetic Operators:

+ - * / % ** //-floor div

2. Assignment Operators:

= += -= *= /= %= //= **= &=
|= ^= >>= <<=

3. Comparision Operators:

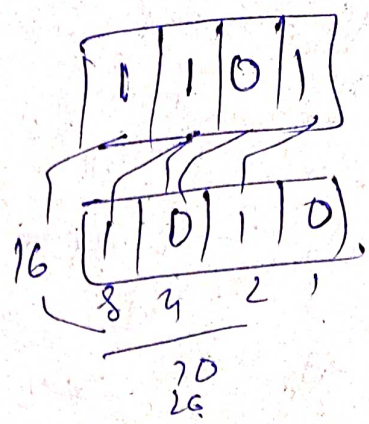
==
!=
>
<
<=
>=

4. Logical Operators:

and

or

not



5. Identity operators:

is

is not

Code is - x = ["aa", "bb"]

y = ["aa", "bb"]

z = x

print(x is z) - True

print(x is y) // FALSE - obj diff

print (x == y) - True.

Code is-not :

x = ["aa", "bb"]

y = ["aa", "bb"]

z = x

print(x is not z) - False

print(x is not y) - True.

print (x != y) - False

6. Membership Operators:

in , not in

Code-1 : x = ["aa", "bb"]

print("bb" in x)

- True.

code-2 : x = ["aa", "bb"]

print("cc" not in x)

- True

7. Bitwise Operators:

& , | ^ ~ >> <<

1.11 Python Functions:

A Function is a self contained block of code which only runs / Executes when we called.

You can pass data, know as parameters, into a function.

A function can return data as a result.

Creating a function:

In Python a function is defined using the "def" Keyword.

Syntax: → def function-name():
// Stmt.

Ex: def my():
print("I am VXM")

Calling a function:

To call a function; use the function name followed by the paranthasis.

Ex: def my():
print("Hai Hello VXM")
my()

o/p: Hai Hello VXM.

1. Function Definition
3. Function Call
2. Function Declaration

(e) Program on Arithmetic Operations using "Function"

Formal parameters

```
def arith(val1, val2):
    print("Addition :", val1+val2)
    print("Subtraction :", val1-val2)
    print("Productum :", val1*val2)
    print("Division :", val1/val2)
    print("Modulation :", val1%val2)
arith(100, 50)
```

Actual parameters.

Exact output :

```
Addition     : 150
Subtraction  : 50
Productum    : 5000
Division     : 2.0
Modulation   : 0
```

Parameters or Arguments.

The terms parameter and argument can be used for the same thing : information passed into a function.

A parameter is a variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it called.

Error Code.

```
def Sum(a1, b1):
    point(a1, b1)
Sum(10)
```

Raise Error

Formal parameters ≠ Actual parameters.

Arguments :

The main purpose is to pass the information to the function. as a arguments.

Arguments are specified after the function name, inside the parantheses. you can add as many arguments as you want, just seperate them with a comma [,]

The following code with the "Single" Argument

Sample Code with "Single Argument"

```
def Sum(nu):
    print(nu)

Sum(10)
```

o/p : 10

Sample Code with "more than one Argument".

(E)

```
def mydetails(name,age, color):
    print("my name is :", name)
    print("my Age is :", age)
    print(" My Color is :", color)

# mydetails(Chandra Sekhar Yalamarthi)
# mydetails(20)
# mydetails(fair).

mydetails("Chandu", 20, "Fair")
```

o/p :

my name is : chandy
my ~~age~~ is : 20
my Color is : fair.

Arbitrary Arguments, *args.

Suppose if you do not know how many arguments that will be passed into your function, add a "*" before the parameter name in the function definition.

This way the function will recive a "tuple" of arguments, and can access the items accordingly.

"Arbitrary Arguments are often shortend to *args in python Document."

Sample Code

(E)

```
def my(*det):
    print("my details:", det[2])
my("Name", "age", "Color", "Bike", "a")
   [0]     [1]     [2]     [3]    [4]
```

Output: color.

Keyword Arguments:

We can also send arguments with the "keys = values" syntax. This way of that order of the arguments does not the matter.

(E)

Code:

```
def sub(m1, m2, m3):
    print(m1, m2, m3)
sub(m2=100, m3=59, m1=25)
```

o/p: 25 100 59

Arbitary Keyword Arguments, **Kwargs.

" The phase Keyword Arguments are often the Shortend to Kwargs in python Documentation".

If you do not know how many keyword arguments that will be passed into you function, add two asterisk: ** before the parameter name in the function definition.

(E) Code:

```
def my(**marks):
    print( marks['a1'], marks['a2'], marks['a3'])
my( a1 = 10, a2 = 20, a3 = 30)
```

Output:

10. 20 30

Default Parameter Value:

If we call the function without argument, it uses the default values:

Code:

(E)

```
def mf( name = "Chandu"):
    print("One of my friend:", name)
    print("One of my friend:", name)
    print("One of my friend:", name)
mf("Abhisheak")
mf("Yugandhar")
mf()
mf("Sagar")
```

o/p:

One of my fried Abhisheak
One of my friend Yugandhar
One of my friend Chandu
One of my friend Sagar.

Passing a List as a Argument:

```
def fruites(fru):
    for x in fru:
        print(x)
fruitesa = ["apple","banana","mango"]
fruites(fruitesa)
```

Output:
apple
banana
mango.

Return Values:

The functions return the return values by using the return statment.

Code:

2 Table upto 5 steps:

```
def tab(n):
    return 2*n
print(1)
print(2)
print(3)
print(4)
print(5)
```

Output:

2
4
6
8
10.

The "Pass" Statment:

If you defines a function. That function is Empty then to avoid the Error we can write the pass statment in it.

Code:

```
def myfun():
    pass
```

Recursion:

1. The process of self calling of a function is known as Recurrion.
2. Python also accepts function recurrion, which means a defined function can call itself.
3. Recurrion is a Common mathematical and programming Concept. It means that a function calls itself. This has a benfit of meaning that you can loop through data to reach a result.
4. The developer should be very Careful with recursion as it can be quite Easy to slip into a writing a function which never terminates, or one that uses Excess amounts of memory or processor power.
5. However, when written Correctly recursion can be a very Efficient and mathematically-elegant approch to programming.

Chandra Tekkali

Syntax:

```
def res():
    // Stmt
    res()
res()
```

Ex: Code on factorial of given number.

```
def fact(x):
    if x==1:
        return 1
    else:
        return (x * fact(x-1))
#num(10)
num = 10
print("factorial is : ", fact(num))
```

Output:

factorial is : 3628800

Python Lambda:

1. A Lambda function is a small anonymous f^n
2. A Lambda function can take any number of arguments, but can only have one Expression.

Syntax:

```
lambda arguments : expression.
```

Sample code to add 10 to argument 'a'.

```
x = lambda a: a+10
print(x(5))
```

o/p: 15

(E)

Lambda function can take any number of arguments

```
x = lambda a, b : a*b
print(x(5,6))
```

op - 30

Usage of Lambda:

1. The power of lambda is better shown when you use them as an anonymous function instead of another function.

2. Say you have a function definition that takes one argument, and that argument will be multiplied with the unknown number.

Code:

```
def myfun():
    return lambda a:a*n

mydo = myfun(2)
print(mydo(11))
```

o/p = 22

[Contd]

→

1.12 Conditional Statments:

The statments which are can be Executed based on "Condition" is known as the Condition Statments:

Statments:

1. Simple if
2. elif
3. else
4. one line if / shorthand if
5. Shorthand if else
6. nested if

1. Simple if:

```
a = 10
b = 20
if b > a:
    print("a is big")
```

o/p: a is big

2. elif:

```
a = 20
b = 20
if b > a:
    print("b is big")
elif a == b:
    print("Both are Equal")
```

o/p: both are Equal

3. else:

```
a = 200
b = 50
if b > a:
    print("b is big")
elif b == a:
    print("Equal")
else:
    print("A Big")
```

o/p: A Big.

4. Short hand if :

```
a=100
b=20
if a>b: print("a is big")
```

o/p: a is big

5. Short hand if...else :

```
a=10,1
b=20
print("A") if a>b else print("B")
```

o/p

B

6. Nested if Stmt :

```
x=41
if x>10:
    print("Above ten,")
    if x>20:
        print(" and also above 20!.")
    else:
        print(" but not above 20.")
```

Output: Above ten,
and also above 20!

1.13 Looping Statments :

The Statments which are can be Excecited a block of Statments untill Some Condition fale

- while loops.
- for loops.

While Loop:

Code-1:

```
i=1
while(i<=6):
    print(i)
    i=i+1
```

OP

1
2
3
4
5
6

Break Statment:

Exit from the loop:

```
i=1
while(i<6):
    print(i)
    # i=i+1
    if(i==3):
        break
    i+=1
```

O/P

1
2
3

Continue Statment:

Skip the data in loop

```
i=1
while(i<6):
    ~~print(i)~~ i+=1
    if(i==3):
        Continue
    print(i)
```

O/P: 1 2 4 5 6

1
2
4
5

else stmt in while Statment.

```
a = 10
while(a<100):
    print(i)
    i+=1
else
    print(" is too hongo ").
```

1 2

do... while loop:

for loop Statment:

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dict, a set or a string).

By list:

```
marks = [100, 98, 75, 65, 30]
for x in marks:
    print(x).
```

100
98
75
65
30

print the single letter in a string.

```
for x in "banana":
    print(x)
```

range()

```
for x in range(6):
    print(x)
```

o/p: 0 1 2 3 4 5

```
for x in range(2,6):
    print(x)
```

o/p: 2 3 4 5

```
for x in range(2,30,3):
    print(x)
```

o/p: 2 5 8

1.14: Modules of Python:

1. Consider a module to be the same as Code library.
2. A file Containing a set of functions you want to include your application.

Creating a Module:

To create a module just save the code you want in a file with the Extension [.py]

Code:

```
def mymod(name):
    print("Hello," + name)
```

] modme.py

- Now we can use the module we just Created, by using the "import" Statment.

Ex:

```
import modme
modme.mymod("Chandu")
```

Variable in the module:

Module Code:

```
person = {
    "name" : "chandu",
    "age" : 20,
    "color" : "fair" }
```

] SAVED BY per.py

Imported here:

```
import per
x = per.person["age"]
print(x)
```

// op 20

Naming a Module:

You can name the module file whatever you like, but it must have the file Extension [.py]

Re-naming a module:

You can Create an [alias] when you import a module by using the [as] keyword.

makes "per" module to "person"

```
import per as person
a = person.mymod["name"]
print(a).
```

Built in Modules:

There are several built-in modules in python which you can import whenever you like.

platform module.

```
import platform
x = platform.system()
print(x)
```

O/p: Windows.

using the dir()

It will display that the all some of the functions that are in platform module.

```
import platform
x = dir(platform)
print(x)
```

Import from Module:

You can choose to import only parts from a module, by using the from keyword.

Example:

The modeled name "mymod" has one the function and one directory.

Code in Module of "mymod"

```
def my(name):
    print("Hello", name)
per = { "name": "Chandu",
        "Age" : 20,
        "Color" : "Black" }
```

```
from mymod import per
print (per["Color"])
```

Output:

Black.

```
a = [1, 2, 3]
try:
```

ZeroDivisionError

NameError

```
except:
```

UNIT-2.

2.1 Iterators:

An Iterator is an object that contains a countable number of values.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values that are present.

Technically in python, an iterator is an object which implements the iterator protocol, which consists of the methods. __iter__() and __next__().

Iterable v/s Iterator:

Iterable:

- is an object, which one can iterate over. It generates an iterator when passed to iter() method.

Iterator:

- is an object, which is used to iterate over. an iterable is an iterator object using __next__(). Iterator has the next() for print the next value.

Note:

Note that every iterator is also a iterable, but not every iterable is an iterator for example, a list is iterable but a list is not an iterator. An iterator can be created from an iterable by using the function iter(). To make this possible, the class of an object needs either a method __iter__ which return an the iterator, or a __getitem__ method with sequential indexes starting with 0.

List, tuples, dictionaries, and sets all iterable objects. They are iterable Containers which you Can get an iterator form.

All these objects have a iter() method which is used to get an iterator.

Example:

Return an iterator from a tuple, & print Each Values:

```
mytup = ("aa", "bb", "cc", "dd", "ee")
myite = iter(mytup)
print(next(myite))
print(next(myite))
print(next(myite))
```

output:

```
aa
bb
cc
```

"Even String as a iterable object, Containing a Sequence of Characters"

Code:

```
mystr = "Chandra Sekhar"
myitr = iter(mystr)
print(next(myitr))
print(next(myitr))
print(next(myitr))
print(next(myitr))
print(next(myitr))
```

output:

```
C
h
a
n
d
```

Loop through an Iterator:

We can also a for loop to iterate through an "iterable" object.

Code-1:

```
mytup = ("apple", "banana", "cherry")
for x in mytup:
    print(x)
```

output: apple
banana
cherry.

Code-2: Ex with a String:

```
mylis = ["aa"] ["bb", "cc"]
for x in mylis:
    print(x)
```

o/p: a
a

Cuts of python for loop:

Iteration:

- The process of loop through the objects or items in a collection.

Iterable:

- An object (or the adjective used to describe an object) that can iterated over.

Iterator:

- The object that produces successive items or values from its associated.

iter() - It's a built-in function used to obtain a iterator for an iterable.

Create an Iterator:

To Create an object/class as an iterator you have to implement the methods __iter__() and __next__ to your object.

As you have learned in the python. classes / objects, all classes have a function called __init__(), which allows you to do the some initalizing when the object being Created.

The __iter__() method acts similar, you can do operations (initalizing etc) but must always return the iterator object itself.

The __next__() method also allows you to do operations, and must return the next item in the sequence.

Code:

```
class Mynum:
    def __iter__(self):
        self.a = 1
        return self

    def __next__(self):
        x = self.a
        self.a += 1
        return x

myclass = Mynum()
myiter = iter(myclass)
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

output:

1
2
3

StopIteration:

The example above would contain forever if you had enough next() statments, or if its was used a for loop.

To prevent the iterations to go on forever, we can use the **StopIteration** statment.

The next() method, we can a terminating condition to raise an error if the iteration is done a specified number of times.

Example:-

```
class Mynum:
    def __iter__(self):
        self.a = 1
        return self
    def __next__(self):
        if(self.a<=20:
            x = self.a
            self.a+=1
            return x
        else:
            raise StopIteration

myclass = Mynum()
myiter = iter(myclass)
for x in myiter:
    print(x)
```

output:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

2.2. Selection if and if-else Statment Condition Iteration To the while Loop.

The process of selection of Simple if and the if-else Condition with the iteration of that to the while Loop.

Using if-else Condition. Statments :

```
n = int (input("Enter no:of terms"))
n1, n2=0, 1
Count = 0
if terms <=0:
    print(" +ve num")
elif nterms == 1:
    print(" Fib Seq upto ", n)
    print(n1)
else:
    print(" Fibonacci Sequence")
    while count <n:
        print(n1)
        nth = n1+n2
        n1 = n2
        n2 = nth
        Count + = 1
```

Output :

```
Enter no:of terms : 7
Fib Seqen upto 7
0
1
1
2
3
5
8
```

using if Condition through while loop :-

```
i = 1
while (i<6):
    i+ = 1
    if(i = = 3):
        break
    i + = 1
```

2.3. Accessing a Character in a Substring :

In python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access to character from the back of a String. Eg -1 refers to the last character, -2 refers the second last character and so-on.

While accessing an index out of the range will cause an "IndexError." Only integers are allowed to the be passed as an index, float or other types will cause TypeError.

C	H	A	N	D	U
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Program :

```
# To print String :
    Stri = "CHANDU"
    print(Stri)        #output : CHANDU
# Point First Character :
    print(String[0])   # C
```

print last character of a String:

```
print(str[-1])   # U
```

print the index of a String.

```
txt = "This is CHANDU"
x = txt.index("is")
print(x)   # 2
```

* * * Substring in python in "Python Strings 1.9" in Unit-II.

8.4 Python Number Systems:

The python Number systems is representing the way of using the below numbers in a python language.

1. Binary Number System - 2
2. Octal Number System - 8
3. Decimal Number System - 10
4. HexaDecimal Number System - 16

Let see one by one, how those are used in the python language.

1. Binary Number Systems:

1. In general, binary number represents 0 or 1 in the System.
2. The base or radix of binary is 2
3. The possible digits that are used in a binary number System are 0 and 1
4. If we wanted to store binary num in python variable that should start with $\boxed{0b}$.

Example:

```
x = 0b1010
print(x)    #    10
```

8 4 2 1
1 0 1 0
8+2 → 10

Note: we can not give the 0b120 since binary will Error only Enter 0 or 1

2. Octal Number:

1. The base or radix of the octal num is 8.
2. The possible digits that are used in the octal number System are 0 - 7.
3. To represent an octal number in python, the number should start with 0 (python2) or 0x (python3)

Example:

```
x = 0123
print('Value:'+x)    # 83
```

zero

Note: only the values in between 0 - 7

3. Decimal Number System:

1. The base or radix of the decimal num is 10
2. The possible digits that are used in the decimal number System are 0 to 9.
3. The default number System followed by python is the decimal number System.

Example:

```
x = 1234
print(x).
```

Note: Value in b/w 0-9 - 10 digits.

4. Hexadecimal Number Systems:

1. The base and radix of the hexadecimal number System 16.

2. The possible digits that are used in hexadecimal number system are 0 to 9 and to f

3. To represent a hexadecimal no in python, the number. Should start with $\boxed{0x}$

Example:

```
x = 0x25
print(x).
```

Output:

37.

2.5 String methods in Text files:

Now you can save or write string to text a file in persistant data storage using "python".

To write string to a Text file, follow these sequence of steps.

1. Open files in write mode using $\boxed{\text{open()}}$ fun
2. Write string to the file using $\boxed{\text{write()}}$ method.
3. Close the file using $\boxed{\text{close()}}$ method.

Example: Write String to Text file.

we will take a string constant and write the string to a text files by following the above said sequence of steps.

Program:

```
tf = open("Samplex.txt", "w")
n = tf.write("welcome to python Example.")
tf.close()
```

Output:

29

Note:

To write() method returns the number of characters written to the text file. Note that this kind of the writing to text file, overwrites the data, if the file is already present. If the file is not present, it creates a new file and then writes the string to the files.

Example 2: Writing other than String to Text file. in Text method.

A file can be opend in two modes: the first one is text mode and the second one is binary mode. By default a file is opend in text mode. However, you can Explicity Specify the mode.

In the following Ex, we will open the file in the text mode by appending "t" to the mode and write the string to a text file by following the sequence of steps mentioned at the starts of this code.

Program:

```
tf = open("Sa.txt", "wt")
n = tf.write(" welcome ")
tf.close()
```

output:

29.

Writing other than String to Text File:

If you would like to write any python object other than String or a bytes object to a file, using write() method, you should first convert that python object to a string or bytes object.

String Operations:

+, *, Concater

```
a = "Hai"
b = "Hello"
c = a + " " + b
print(c)
```

addition

```
cost = 15;
a = " Cost is : ";
c = a + str(cost);
print(c)
```

multi:

```
a = 'hip';
b = a * 2 + 'hurry!'
print(b * 3)
```

UNIT - 2

3.1 Python Collections:

python Allows 4 python Collection datatypes are the.

1. List
2. Tuple
3. Set
4. Dictionary.

List:

List items are ordered, changeble & allow duplicate values.

Tuple:

Tuple are ordered, unchangeble, and allow duplicate values.

Sets:

Set items are unordered, unchangeable and do not allow duplicate values.

Dictionaries:

Dictionary are unordered, changeable and does not allow duplicates.

SNO	Collections	ordered / unordered	changable / unchangable	allow / not allow duplicates
1.	Lists	ordered	changable	allow duplicates
2.	Tuples	ordered	unchangable	allow duplicates
3.	Sets	un-ordered	unchangable	not allow duplic
4.	Dict	un-ordered	changable	not allow duplicates.
	numpy	ordered	unchangable	

3.1.1 Python Lists:-

Lists are used to store multiple items in a single variable

Lists are created by "[ ]".

Creating a List:

```
a = [1, 2, 3, 4, 5]
print(a)
```

Lists allow Duplicates:

```
a = [1, 2, 3, 2, 3, 1]
print(a)
```

o/p: [1, 2, 3, 2, 3, 1]

List Length:

```
a = [1, 2, 3]
print(len(a))
```

o/p: 3

List items DataTypes:

```
a = [1, 2, 3]
b = ['1', '2', '3']
c = [True, False]
print(a)
print(b)
print(c)
```

o/p = [1, 2, 3]
['1', '2', '3']
[True, False]

List Contain different Datatypes:

```
a [ 1, "aa", 'Rama', True, 4.5]
print(a)
```

o/p:

[1, "aa", 'Rama', True, 4.5]

List type: by type()

```
a = [1, 2, 3]
print(type(a))
```

o/p : <class 'list'>

list() - Constructor (( )) - rounded - double parranthasis.

```
a = list(("aa", "bb", "cc"))
print(a)
```

o/p : ['aa', 'bb', 'cc']

Access List items :

```
a = [1, 2, 3]
print(a[0])
```

o/p = 1

Negitive indexing :

```
a = [1, 2, 3, 4, 5]
print(a[-2])
```

o/p : 4

Range of Indexes :

```
a = [0, 1, 2, 3, 4, 5]
print(a[2:5])
```

Start index - include
End index - Exclude

o/p : [2, 3, 4]

```
print(a[:5])
```

o/p : [0, 1, 2, 3, 4]

```
print(a[2:])
```

o/p : ['2', '3', '4', '5']

```
a = [1, 2, 3, 4, 5]
print(a[-4 : -1])
```

o/p : a = [2, 3, 4]

Change List items:

```
a = ["aa", "bb", "cc"]
a[1] = "dd"
print(a)
```

O/p : ['aa', 'dd', 'cc']

Change Range item values:

```
a = ["aa", "bb", "cc", "dd", "ee", "ff"]
a[1:3] = ["gg", "kk"]
print(a)
```

O/p : ['aa', 'gg', 'kk', 'dd', 'ee', 'ff']

Insert items:

- insert()

```
a = ["aa", "bb", "cc"]
a.insert(2, "dd")
print(a)
```

O/p : ['aa', 'bb', 'dd', 'cc']

Add List items:

- append()

```
a = ["aa", "bb", "cc"]
a.append("dd")
print(a)
```

O/p : ['aa', 'bb', 'cc', 'dd']

Insert Items: # see before page.

Extend List:

```
a = ["aa", "bb", "cc"]
b = [1, 2, 3]
a.extend(b)
print(a)
```

o/p: ['aa', 'bb', 'cc', 1, 2, 3]

Add any iterable:

Add list to tuple:

```
a = ["aa", "bb", "cc"]
b = [1, 2, 3]
a.extend(b)
print(a)
```

o/p: ["aa", "bb", "cc", 1, 2, 3]

Remove List Items:

- remove()
- pop() - ~~not~~ for specific Element defau last ele
- del. - Complete List

remove

```
a = ["aa", "bb", "cc"]
a.remove("bb")
print(a)
```

o/p: ["aa", "cc"]

```
a.pop()
```

print(a) - o/p: ['aa']

del a[0] print(a) o/p: [ ]

del :

```
a = ["aa","bb","cc"]
del a
```

o/p :

Clear the list

```
a = [1, 2, 3]
a.clear()
print(a)
```

o/p : [ ]

Loop Lists :

```
a = [1, 2, 4, 5]
for x in a:
    print(x)
```

o/p : [1, 2, 4, 5]

Loop through index :

- range()
- len()

```
a = [1, 2, 3, 4, 5]
for x in range(len(a)) :
    print(a[x])
```

o/p :
1
2
3
4
5

Loop using List Comparision.

```
a = [1, 2, 3]
[ print(x) for x in a]
```

o/p :
1
2
3

Sort List :

- Sort()

```
a = [1, 2, 3]
a.sort()
print(a)
```

o/p : 2, 3, 11

Ascending Order

Descending Order :

```
a = [1, 2, 3]
a.sort(reverse = True)
print(a)
```

o/p : 3
2
1

Customize Sort Function :

```
def my(n):
    return abs(n-50)
a = [100, 50, 65, 82, 23]
a.sort(key = my)
print(a)
```

o/p : [50, 65, 23, 82, 100]

Case Insensitive Sort - Sort()

Reverse Order :

```
a = ["aa", "bb", "cc"]
a.reverse()
print(a)
```

o/p : ['cc', 'bb', 'aa']

Copy List :

- copy()
- list()

1. copy() :

```
a = ["aa", "bb", "cc"]
b = a.copy()
print(b)
```

o/p : ['aa', 'bb', 'cc']

2. list() :

```
a = [1, 2, 3]
b = list(a)
print(b)
```

o/p : [1, 2, 3]

Join Lists :

1. Join two Lists :

```
a = ["a", "b", "cc"]
b = [1, 2, 3]
c = a + b
print(c)
```

o/p : ['a', 'b', 'c', 1, 2, 3]

2. append() "

```
a = [1, 2, 3]
b = ["a", "b", "c"]
for x in b:
    ~~a.list~~
    a.append(x)
print(a)
```

2. Tuples:

Tuples are used to store multipl. items in a single Variable.

It is denoted by parathesis "( )".

✓ unchangable

✓ Ordered

✓ Allows Duplicate

```
a = (1, 2, 3)
print(a)
```

o/p: (1, 2, 3)

1. Tuple Length:

```
a = (1, 2, 3)
print(len(a))
```

*. Note:

1.
```
a = ("aa",)
print(type(a))        # <class 'tuple'>
```

2.
```
a = ("aa")
print(type(a))        # <class 'str'>
```

Tuple Constructor:

```
a = tuple ("aa", "bb", "cc"))
print(a)
```

o/p: ('aa', 'bb', 'cc')

Access Tuple Items :

```
a = (1, 2, 3)
print(a[1])
```

o/p : 2

```
print(a[-1])
```

o/p : 3

```
print(a[1:2])
```

o/p : 1

Update Tuples :

Convert Tuple → List

```
a = ("aa", "bb", "cc")
b = list(a)
b[1] = "dd"
a = tuple(b)
print(a)   // tuple
b.remove("bb")
print(b)   // remove
b.append("dd")
print(b)   // add.
del a
print(a)   // ( )
```

Unpack Tuples:

When we create a tuple, we normally assign values to it. This is called "packing" a tuple.

Packing a tuple:

```
a = ("a", "b", "c")
print(a)
```

o/p: ('a','b','c')

Unpacking a tuple:

```
a = ("aa", "bb", "cc")
("dd", "kk", "hh") = a
print(dd)
print(kk)
print(hh)
```

o/p: aa
bb
cc

Using *:

```
a = (1, 2, 3, 4, 5)
# (a, b, c, d, e, f) = a
# print(
(a, b, *c) = a
print(a)
print(b)
print(c)
```

o/p: 1
2
[3, 4, 5]

Join Tuples:-

Ex-1

```
a = (1, 2, 3)
b = (4, 5, 6)
c = a+b
print(c)
```

Ex-2:

```
a = (1, 2, 3)
b = a*5
print(b)
```

o/p: (1 2 3 1 2 3 1 2 3 1 2 3 1 2 3)

III Python Sets :

Unordered.
Unchangable.
Doesn't allow Duplicates. by { }

Create a Set :

```
a = {"aa", "bb", "cc"}
print(a)
```

Ex: {'aa', 'bb', 'cc'}

Set Constructor :

set()

```
a = set(("aa", "bb", "cc"))
print(a)
```

o/p : {'aa', 'bb', 'cc'}

Accessing set Items in a List :

By Loop:

```
a = {"aa", "bb", "cc"}
for x in a:
    print(x)
```

o/p : aa
bb
cc

Checking:

```
a = {"aa", "bb", "cc"}
print("bb" in a)
```

o/p : True.

Add Items:

- add()

```
a = {"aa", "bb", "cc"}
a.add("dd")
print(a)
```

o/p: {'aa', 'bb', 'cc', 'dd'}

Add Sets: - update

Set B is added to set A

```
a = {"aa","bb","cc"}
b = {"dd", "ee", "ff"}
a.update(b)
print(a)
```

o/p: {'aa', 'bb', 'cc', 'dd', 'ee', 'ff'}

Add any Iterable:

```
a = {"aa", "bb", "cc"}
b = [1, 2, 3]
a.update(b)
print(a)
```

Output: {'aa', 'bb', 'cc', 1, 2, 3}

Remove items:

- remove()
- discard()
- del
- clear()
- pop()

```
a = {"aa", "bb", "cc", "dd", "ee"}
a.remove("cc")
print(a)          // {'aa', 'bb', 'dd', 'ee'}
a.discard("bb")
print(a)          // {'aa', 'dd', 'ee'}
x = a.pop()
print(x)          // {'aa', 'dd'}
a.clear()
print(a)          // set()
del a
print(a)          // raise an Error
```

Join Sets:

- union()
- update()

Ex: union()

```
a = {"a", "b", "c"}
b = {1, 2, 3}
c = a.union(b)
print(c)
```

output:

{"a", 1, "b", "c", 2, 3}

Ex: update()

```
a = {"a", "b", "c"}
b = {1, 2, 3}
a.update(b)
print(a)
```

o/p:

{1, "a", "b", 2, 3, "c"}

IV. DICTIONARY

It Stores the keys and the values.

Create:

```
a = { "name" : "Chandu",
      " age " : 20,
      " color" : "fair"}
print(a)
```

o/p: {'name': 'Chandu', 'age': 20, 'color': 'fair'}

Access Dictionary Elements:

```
a = {"a": 1 , "b": 2, "c": 3}
print( a[b])
```

o/p: 2

Access by using key

using get()

```
a = {"a": 1, "b": 2, "c": 3}
x = a.get("a")
```

o/p: 1

Get keys:

```
a = { "a": 10,
      "b": 20,
      "c": 30} #print(a.gets(
print(a.keys())
```

o/p: dict_keys(['a', 'b', 'c'])

Add item to the "DICT"

```
a = { "a" : 10,
      "b" : 20,
      "c" : 30 }
x = a.keys()
print(x)
a["d"] = 40
print(a).
```

Get Values:

```
a = { "a" : 10,
      "b" : 20,
      "c" : 30}
print(a.values())
```

o/p : dict_values([10, 20, 30])

Change Dict Values : - update

```
a = {"a" : 10,
     "b" : 20,
     "c" : 30 }
a.update({"c" : 40})
```

o/p : {'a' : 10, 'b' : 20, 'c' : 40}

Add Items:

```
a = {"a" = 1,
     "b" = 2,
     "c" = 3}
a["d"] = "4"
print(a)
```

o/p

{'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4}

Remove Items:

- remove()
- pop()
- clear()
- pop()
- popitem()

```
a = {"aa": 10,
     "bb": 20,
     "cc": 30,
     "dd": 40,
     "ee": 50,
     "ff": 60}
a.pop("bb")
print(a)   # {"aa": 10, "cc": 30, "dd": 40, "ee": 50, "ff": 60}
a.popitem()
print(a)   # {"aa": 10, "cc": 30, "dd": 40, "ee": 50}
a.clear()
print(a)   # { }
del a
print(a)   # raise error because it deleted & display.
```

Copy Dictionary:

```
a = {"aa": 10,
     "bb": 20,
     "cc": 30}
x = a.copy()
print(x)
print(a)
```

o/p:

{"aa": 10, "bb": 20, "cc": 30}

{"aa": 10, "bb": 20, "cc": 30}

Nested Dictionaries :

```
# myfamily = {#
    "mom": {
            "Name": "Chandu",
            "Age": 36
        },
    "DAD": {
            "Name": "Raj",
            "Age": 47
        },
    "SIS": {
            "Name": "Yuva",
            "Age": 16
        },
    "ME": {
            "Name": "Sekhar",
            "Age": 20
        }
```

Output :-

```
{ 'MOM': { 'name': 'chu
           'age': 36
         },
  'DAD': { "name": 'Raj'
           age: 36 },
  "SIS": { "name": 'Yuva'
           age: 16
         },
  "ME": { 'name': 'Sekhar'
          'age': 20 }
```

```
Yalamarthi = {
    "mom" : mom
    "DAD" : DAD
    "SIS" : SIS
    "ME" : ME
}
```

3.2 Defining a Simple Function with Lists:

```
def subindip():
    print("List of programming Subjects
          that are in Diploma CME")
    print()
    print()
    a = ['C-Lang',
         'Java',
         'DS-C',
         'MAD',
         '.Net']
    print(a)

subindip()
```

Output

List of programming subjects that are in Diploma CME

['C-Lang', 'Java', 'DS-C', 'MAD', '.Net']

Defining a Simple Function with Dictionaries.

```
def Marks():
    sub = {"101": 90, "102": 96, "103": 78,
           "104": 98, "105": 94}
    total = sub["101"] + sub["102"] + sub["103"]
            + sub["104"] + sub["105"]
    print("Total :", total)
    print("Average :", (total/5), "%")

Marks()
```

Output:

Total : 456

Average : 91.2 %

3.3 Functions as a Abstraction :

Through it is very simple, sum-squares exemplifies the most powerful property of user-defined functions.

The function sum-squares is defined in terms of the function square, but relies only the relationship that squares defines between its input arguments and its output values.

We can write sum-square without concerning ourselves with how to square a number. The details of how the square is computed can be supressed, to be considered at a later time. Indeed, as far as sum-squares is concerned a square is not a particular function body, but rather an abstraction of a function, a so called functional abstraction. At this level of abstraction my function that computes the square is equally good.

Thus, considering only the values they return, the following two functions for square a number should be indistinguishable. Each takes a numerical argument and produces the square of that number as the value.

Code:

```
def square(x):
    return mul(x, x)
def square(x):
    return mul(x, x-1) + x
```

In other words, a function definition should be able to supress details. The users of the function my not have written the function themselves, but may have obtained it from another programmer as a "black box". A user should not need to know how the function is implemented in order to use it. The python library has this property. Many developers use the functions defined users there. but few ever inspect their implementation. In fact; many of the implementations of python library functions are not written in python at all, but insted in the C-lang.

Abstract Class in python:

An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has declaration but does not have an impentation while we are designing large function units we use an abstract class. when we want to provide a common interface for different implementations of a component, we can use the abstract class.

Why we use AbstractBaseClasses: "ABC".

By defining a abstract base class, you can define a common Application program Interface (API) for a set of sub classes. This is capability is especially useful in situations where a third-party is going to provide implementation

Such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

How AbstractBaseClasses work:

By default, python does not provide abstract classes. python comes with a module which provides the base for defining [ABC] AbstractBaseClasses and that module name is $\boxed{\text{ABC}}$. ABC works by decorating methods of the Base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword. @abstractmethod.

Code Example:

```
from abc import ABC, abstractmethod
class polygon(ABC):
        def noofsides(self):
                pass
class Triangle(polygon):
        def noofsides(self):
                print(" 3 Sides ")
class Circle(polygon):
        def noofsides(self):
                print(" 0 Sides ")
```

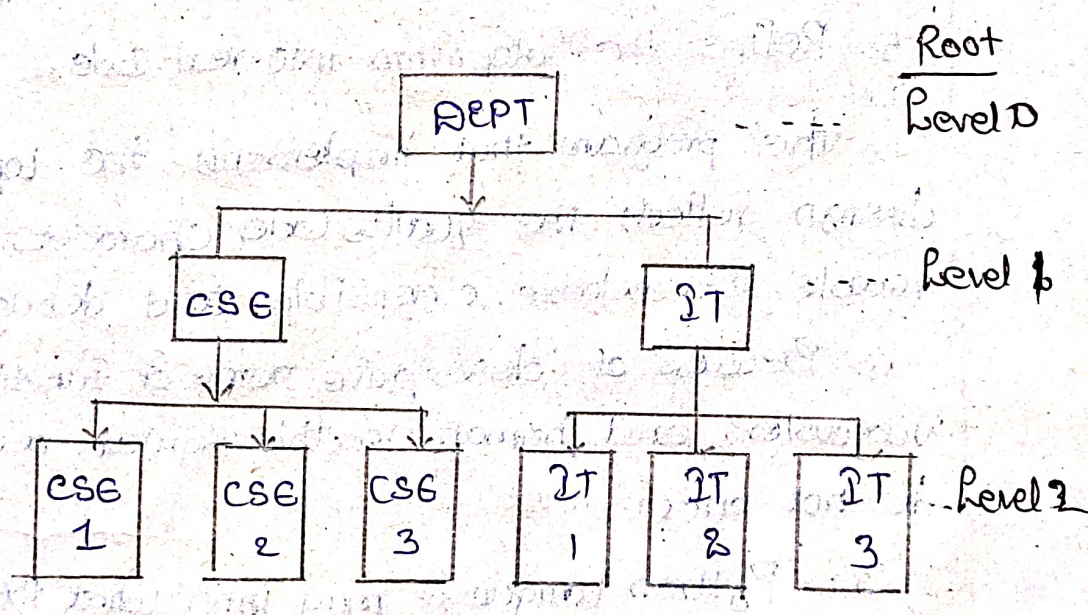
```
# p = polygon()
t = Triangle()
t.noofsides()
c = Circle()
c.noofsides()
```

Output:

3 Sides
0 Sides.

3.4 Problem Solving with Top-Down Design.

The Top-down design approch is to produce a plan for solving a complicated problem. The output of this design is a tree-like structure chart as shown below. Consists on 3 Levels.



The root entry point of the program. The first level contains the major subtasks. These subtasks work together to solve the initial problem. If a subtask is not trivial at the first level, we divide it into smaller tasks, and then we place all these similar tasks at the second level. This process continues will we reach a level at which breakdown is not necessary. These are

Several levels between the root & leaves. At Each level, the dividable subtask reference subtasks at the lower level. we define interfaces (or signatures) of these subtasks to communicate with other subtasks at the different levels. we use the following procedure to walk through the top-design Approch.

procedure steps :

1. Start with an initial problem statment
2. Define subtask at the first level.
3. Divide subtasks at a higher level into more specific tasks.
4. Repet step(3) untill Each subtask is trivial.
5. Refine the algorithm into real code.

1. The program that implements the top-down design reflects the quality code characteristics : reable, extendable, extensible, and debuggable.

2. Because of descriptive name of functions & variables, and human-readble syntax, people who do not know.

3. Python language may infer what the program is doing. when we have new requirements, we can add a piece of code without modifying existing functions.

4. If we find program does not work Correctly we only need to run suspensious function to locate the problematic function. Besides, this program allows other people to improve one function without touching code in other functions.

5. Top-down design and object oriented program (OOP) can co-exist.

6. The top-down design approch answers a question about how to start tackling a complicated problem.

7. However, the top-down design approch has some limits :

8. If we do not divide a task into several independent subtasks carefully we may face a challenge when implementing those subtasks at the lower levels.

Key points :

1. Basic - Breaks the massive problem into smaller subproblems.
2. Process - Sub modules are solitarily analysed
3. Communication - Not required in the top-down approch
4. Redundancy - Contain redundant information.
5. Programming - Structural / Procedural oriented programming languages (i.e C) follows the top-down approch
6. Mainly used in - Module documentation, test case creation, code implementation and debugging.

3.5 Modules : See the 1.14 Concept in Unit-1

Standred Modules :

There are 4 modules are previously defined that are the - mod1.py
- mod2.py
- mod3.py
- mod4.py.

200 modules.- The python Standred modules library Contains well over 200 modules, although the exact number varies between distrubtion.

There are 4

#Stan 1

1. Statistic Modules
2. Random Modules
3. Collection Modules.
4. Math Modules

1. Statistics Module :

This module is mention in python 3

1. mean()
2. median()
3. mode()
4. median_low()
5. median_high()
6. variance()
7. stdev()

Code:

```
import statistics
A = [5,10, 6, 21, 13,17, 14, 8, 3,9]
mean = statistics.mean(A)
mode = statistics.mode(A)
median = statistics.median(A)
median_low = statistics.median_low(A)
median_high = statistics.median_high(A)
```

```
variance = statistics.variance(A)
std = statistics.stdev(A)
print(" mean =", mean)
print(" mode =", mode)
print("median =", median)
print("median-low =", median-low)
print("median-high =", median-high).
print(" variance =", variance)
print("std =", std)
```

Output:

```
mean     : 10.6
mode     : 5
median   : 9.5
std      : 5.64111
variance : 31.822
Mean-low : 9
Mean-high : 10
```

2. Math Module:

sqrt()	factorial()	log(x, base)	
ceil()	exp()	pi	radians()
floor()	pow(base, x)	degree(pi)	

Code:

```
import math
a=10
print("Sqrt()", math.sqrt(a))
print("fact()", math.factorial(a))
print("floor()", math.floor(a))
print("ceil()", math.ceil(a))
print("Exp", math.exp(a))
print("rad()", math.radians(a))
```

```
print("power()", math.pow(10,a))
print("log()", math.log(a,10)
print("pi ", math.pi)
print("degree()", math.degrees(math.pi/2))
```

Output:

```
sqrt()    - 3.1622
fact()    - 3628800
ceil()    - 10
floor()   - 10
exp()     - 22026.465
rad()     - 0.17453
pow()     - 10000000000.0
log()     - 1.0
pi        - 3.1415926
degrees() - 90.0
```

3. Random Module:

randint(a,b)	shuffle(x)	gauss(mu, sigma)
choice(seq)	random()	uniform(a, b).

Code:

```
import random
print("uniform()", random.uniform(5.5,10.5)
print("gauss()", random.gauss(5,2))
print("random()", random.random())
x = list(range(1,11))
random.shuffle(x)
print("shuffle :", x)
print("choice()", random.choice(['H','T']))
print("randint()", random.randint(1,11)
```

Output :

9.03677
4.738
0.948
T
7
Shuffled List : [9, 1, 10, 6, 2, 8, 3, 4, 7, 5]

4. Collection Methods:

list()
tuple()
set()
dict()

Python Packages :

Package:

packages are way of structuring many packages and modules which helps in well-orginized hierarchy of data set, making the directories and modules easy to access. Just like there are different drives and folders in as OS to help us store files, Similarily packages help us is storing other sub-packages and modules, so that is used by the user when necessary.

Creating & Exploring Package:

To tell python that particular directory is a package, we create a file named-__init__.py inside it and then is considered as a package & we may create other modules and sub-packages within it. This __init__.py file can be left blank or can be coded. with the initialization code for packa

Steps for Create package:

1. First, we create a directory and give it a package name, preferably related to its operation.
2. Then we put the classes and the required fun in it.
3. Finally we create an --init--.py file inside the directory, to let python know that the directory is a package.

Example:

Lets create a package name "Cars" and build three modules in it namely, BMW, Audi & Nissan.

* First we create a directory and name it Cars.
* Then we need to create modules.
  - To do this we need to create a file with the name "BMV.py" and create its content by putting this code into it.

Code for BMV.py

```
class BMV:
    def __init__(self):
        self.models = ['i8', 'x1', 'x5', 'x6']
    def outModels(self):
        print("Available in BMW")
        for model in self.models:
            print(models)
```

Code: Audi.py

```
class Audi:
    def __init__(self):
        self.models = ['q7','q6','q8','q3']
    def outModels(self):
        print("Available in Audi")
        for model in self.models:
            print(model)
```

Code: Nissan.py:

```
Class Nissan:
    def __init__(self):
        Self.models = ['altima','370z','Cube',
                       'rogue']
    def outModels(self):
        print("Available in Nissan")
        for model in self.models:
            print(model)
```

Car.py

3. Finally we create the __init__.py file:

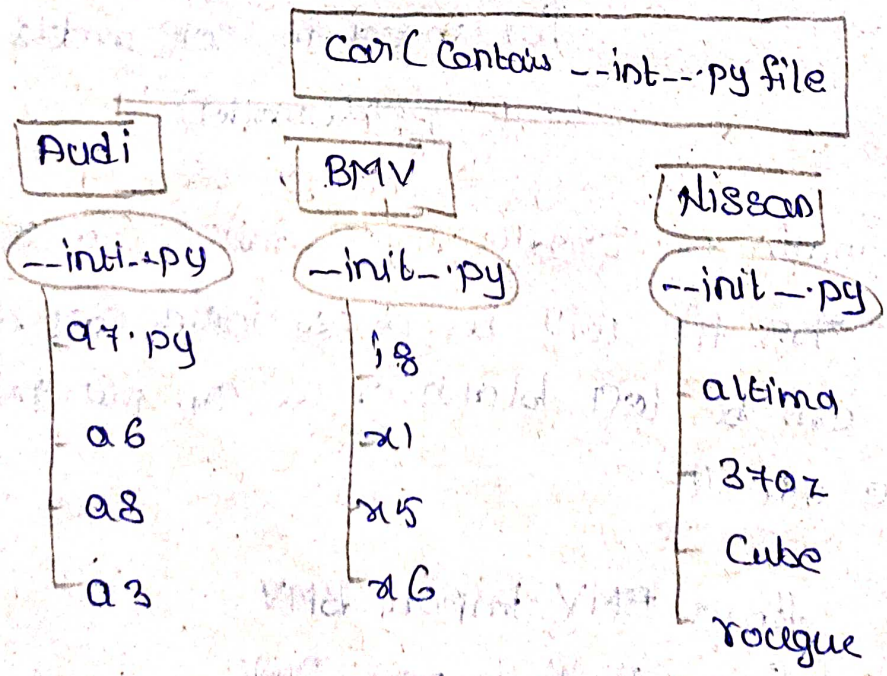
This file will be placed inside Cars directory and can be left blank or we can put this initialisation code into it.

```
from BMV import BMV
from Audi import Audi
from Nissan import Nissan
```

Now, lets use the package that we created. To do this make a simple.py file in the same directory where Cars package is located and add the following to it.

```
from Cars import BMV
from Cars import Audi
from Cars import Nissan

ModBMV = BMV()
ModBMV.outModels()

ModAudi = Audi()
ModAudi.outModels()
# ModAudi
ModNissan = Nissan()
ModNissan.outModels()
```

Various ways of Accessing the packages



1. import in Packages:

```
import Cars.Bmw.x5
```

If any module contains a function and we want to important that for E.g, a8 has a fun get-buy(1) and we want to import that, the Syntax would be.

```
import Cars.Audi.a8
Cars.Audi.a8.get-buy(1)
```

While using just the import Syntax, one must keep in mind that the last attribut must be a sub package or a module, it should not be any functions or class-name.

2. from ... import in packages.

Now, whenever we requrie using such function we would need to write the whole long line after importing the parent package. To get through this in a similar way we use 'from' keyword. For this we first need to bring in the module using 'from' and 'import'.

```
from Cars.Audi import a8
a8.get-buy(1)
from Cars.Audi.a8 imort get-buy.
get-buy(1).
```

3. from import * - all

```
from Cars.Chevrolet import *
```

Numpy Array :

1. ndarray, an efficient multidimensional array, providing fast array-oriented, arithmetic operations and the flexible broadcasting capabilities.

2. Mathamatical functions are for fast operations on entire arrays of data without having to write loops.

3. Tools for reading / writing array data to disk and working with memory mapped files.

4. Linear algebra, random number generation, and fourth fourier transform capabilities.

5. A C API for connecting Numpy with libraries written in C, C++, or FORTRAN.

Advantage :

- Fast Accessing.

6. For most data Analysis applications, the main areas of functionality to focus on are.

7. Fast vectorized array operations for data munging and cleaning & subsetting and that filtering, transformation, and any other kinds of computations.

8. Common array algorithms like sorting, unique and set operations.

9. Effecient descriptive statistics and aggregating summarizing data.

10. Data alignment & relational data manipulation for merging and joining together hetrogeneous datasets.

11. Expressing Conditional logic as array expressions insted of loops with if-elifelse branches.

12. Group-wise data manuplications (aggregation, transformations, function application).

Pandas Basic is Numpy

Why Numpy :

1. Numpy internally stores data in a contignous block of memory, independent of other built-in python objects. Numpy's library of algorithms written in the C-language can operate on this memory without any type checking or other overhead. Numpy arrays also use much less memory than built-in python sequences.

2. Numpy operations perform complex computations on entair arrays without the need for python for loops.

3. Numpy-based Algorithm are generally 10-100 times faster (or more) than their pure python counterparts and use significantly less memory.

- Numpy is a "View".

Object Oriented Concepts in Python:

Object oriented programming is a way of Computer programming using the idea of "objects" represents data and methods. It is also an approch used for Creating neat and reusable Code insted of redundant one. The program is divided into self Contained objects or several mini programs. Every individual object represents a different part of the application having its own logic and data to Communicate within themselves.

Comparision B/w OOP and POP.

OOP	POP
1. It is a bottom-up approch	1. It is a top-down approch.
2. program divides into objects.	2. program divides into functions
3. It is more Secure	3. It is less Secure.
4. Object Can move freely within member functions	4. Data Can move freely from function to function with programs.
5. It Supports Inheritance	5. It Not support Inheritance
6. Access modifiers public private protected.	6. Doesn't use any access modifiers
7. Ex: C++ Java. Algol python.	7. Ex: APP C Java

OOP's Concepts in python:

major OOPS Concepts in OOP's of python is includes class, object, method, inheritance, polymorphism, Data Abstraction & Encapsulation.

Class and Object:

Class:

A class is a collection of objects or you can say it is a blueprint of objects defining the common attributes and behaviour. Now the que arises, how do you do that?

well, it logically groups the data in such a way that code resubility becomes easy. Suppose there is a Class "Employee" and the attributes are ename, esal, eid and eage are the obj in python. Let us see in coding.

class is defined using "class" keyword.

Syntax:

class-keyword class-name (parameter list):

Ex: class Sum( ):

python is not case-sensitive.

Objects:

objects are an instance of a class. It is an entity that has state and behaviour. In a nutshell, it is an instance of a class that can access the data.

Syntax :

object-name = class-method()

Ex:

obj = Stem()

Here "obj" is the object of class

Simple Example:

```
class my():
    def sum():
        print(" my class")

s = my()
s.su()
```

Arithmetical Operations Code:

```
class Arith():
    def add(a,b):
        print("sum =", (a+b))
    def sub(a,b):
        print("sub =", (a-b))
    def mul(a,b):
        print(" mul =", (a*b))
    def div(a,b):
        print(" div =", (a/b))

ad = Arith()
ad.add(20,10)
ad.sub(20,10)
ad.mul(20,10)
ad.div(20,10)
```

output:

```
sum = 30
sub = 10
mul = 200
div = 2.0
```

Object oriented Methodologies :

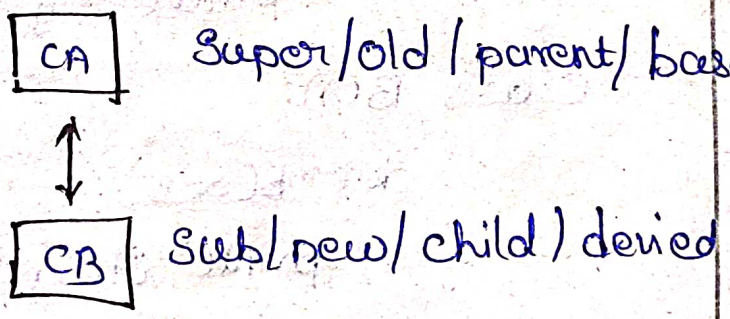
The OOM Concepts are under below.

1. Inheritance
2. polymorphism
3. Encapsulation
4. Abstraction.

1. Inheritance :

The process of deriving a new class from already existing class is known as Inheritance. In generally means "inheriting or transfer of characteristics from the parent class to the child class without any modifications.

The new class is also known as a derived class/child class/new and one from which is derived is called parent/base/old class



Types of Inheritances :

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchal Inheritance
4. Multiple Inheritance.
5. Hybrid Inheratance.

1. Single Inheritance:

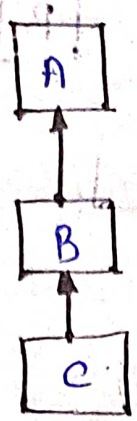
one super class, one sub class

program:

```
class A():
    def a(self):
        print("Super class");
        print("old class")
        print("Parent class")
        print("My obj=", b)
class B(A):
    def b(self):
        print("Sub class")
        print("new class")
        print("Child class")
        print("My obj:", b)
b = B()
b.b()
b.a()
```

2. Multilevel Inheritance:

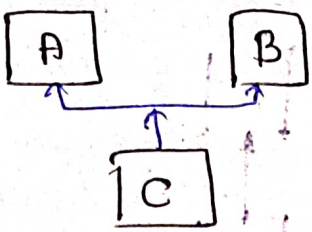
one super class and sub-superclass & subclasses



Program:

```
// class A(self):
    class A():
        def a(self):
            print("Super Class")
    class B():
        def b(self):
            print("Sup/Sub class")
    class C():
        def c(self):
            print("Sub class")

c = C()
c.a()
c.b()
c.c()
```

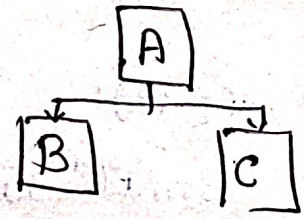
3. Multiple Inheritance:



Program:

```
class A():
        def a(self):
            print("Super Class A");
class B():
        def b(self):
            print("sup/sub class B")
class C(A, B):
        def c(self)
            print("Sub class")
c=C()
c.a() c.b() c.c()
```

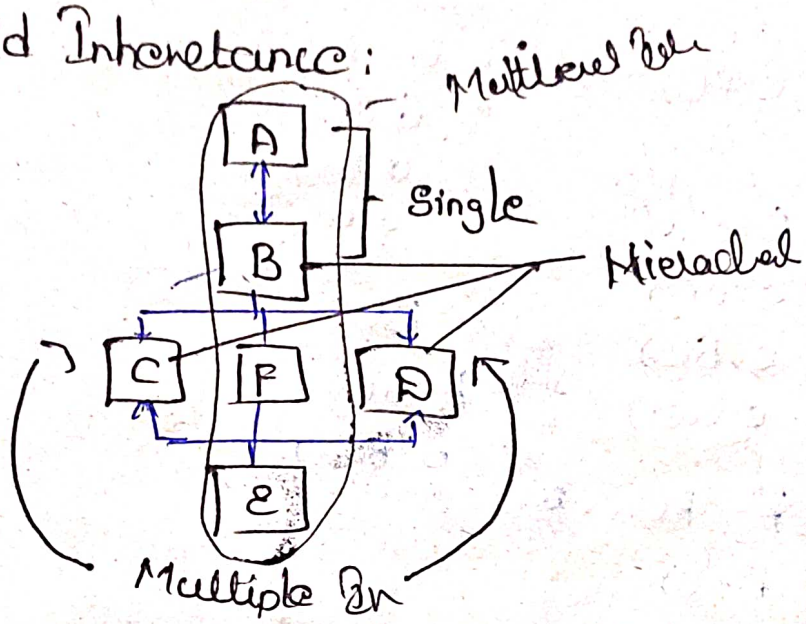
4. Hierarchal Inheritance:




```
class A():
        def a(self):
            print("class A")
class B(A):
        def b(self):
            print("class B")
class C(A):
        def c(self):
            print("class C")
```

```
b = B()
c = C()
b.a()
b.b()
c.a()
c.c()
```

5. Meibrid Inheretance:



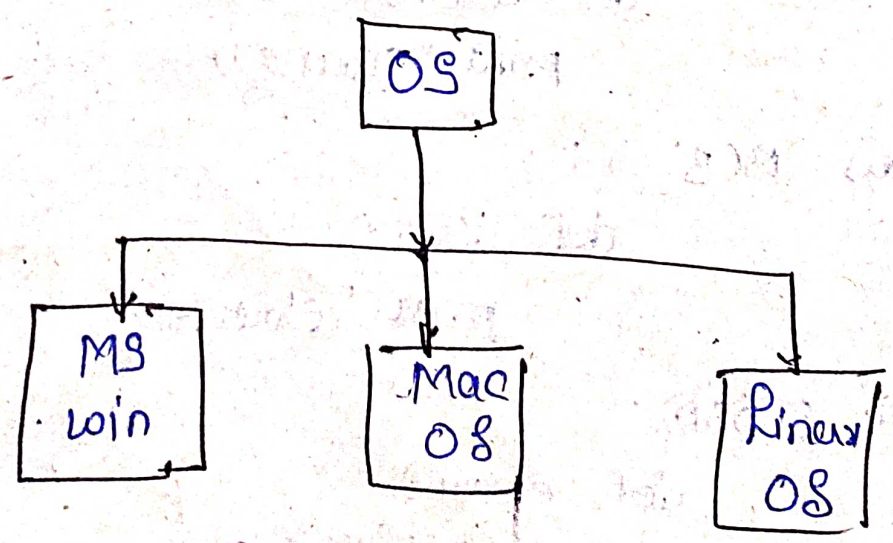
Code:

```
class A( ):
        def a(self):
            print("Class A")
class B( ):
        def b(self):
            print("Class B")
class C(B ):
        def c(self):
            print("Class C")
class D(B ):
        def d(self):
            print("Class D")
class E(C,F,D):
        def e(self):
            print("Class E")
```

```
class F(B):
        def f(self):
            print("class F")

e = E()   // multiple
e.e()
e.c()
e.f()
e.d()

d = D()
d.d()
d.b()
d.a()

c = C()
c.c()
c.b()
c.a()

b = B()
b.a()
b.b()

f = F()
f.f()
f.a()
f.b()
```

Polymorphism :

The ability to take more than one form that is suppose we consider the Operating System that are in the forms of the MS windows, MacOS, Linux and Etc.



Two types of polymorphism:

1. Compile-time / method overloading
2. Run-time / method overriding.

1. Method overloading:

Same method name. diff parameters.
The method overloading will execute only the latest method that we defined.

Code: using dispatch

```
from multipledispatch import dispatch
class A():
    @dispatch(int, int)
    def sum(a, b):
        print("sum =" + (a+b))
    @dispatch(int, int, int)
    def sum(a,b,c):
        print("sum =", (a+b+c));

dis = A()
dis.sum(10, 20)
dis.sum(10,20,30)
```

2. Method overriding:

In the two different classes the method have same name and same parameters.

Code:

```
class A():
        def sum(a, b):
                print("Sum =", (a+b))
        def mul(self, a, b):
                print("Mul =", (a+b))

class B(A):
        def sum(self, a, b):
                print("Sum", (a+b))
        def mul(self, a, b):
                print("Mul", (a*b))

a = A()
b = B()
b.sum(20, 30)      # 30
a.sum(10, 20)      # 30.
```

Using properties to Control Attribute access

```
class Circle(object):
    def __init__(self, radius):
        self.radius = radius
    @property
    def radius(self):
        print 'Get'
        return self.__radius
    @radius.setter
    def radius(self, radius):
        print 'Set'
        assert radius > 0, "The radius must be nonzero and positive"
        self.__radius = radius
print Creat a circle
c = Circle(2)
print "modify circles"
c.radius = 3
print 'verify'
print c.radius
```

modules : num , shell , glob, re , requests.

num modules :

In this num modules provides numeric and mathrelated functions and data types. then number modules defins the hieracy of numeric types.

The math and cmath contains various mathematical function for floating point and the Complex Number. The decimal modules supports exact represention of decimal numbers, using arbitary precision arithmetic.

number modual :

Complex :

- real
- img

Rational.

- numerator
- denominator

Real :

round()
math. floor()
math. ceil()

Decimal Module :

Its a package.

import - module

from - packages

Math module :

math.ceil(x)
math. copysign(x,y)
math. fabs(x)
math. factorial(x)
math. floor(x)

c-math :

cmath. exp(x)
cmath. log(x)
cmath. log10(x)
cmath. sqrt(x)
cmath. acos(x)
cmath. asin(x)

Shutil module: In python provides many (of high-level operations on files are collection of files. It comes under pythons standard utility modules. This modules helps in automatic process of copying. are removal of files and dict.

main purpose: copy:

Syntax:

shutil.copy (source, destination, *, follow_symlinks = True)

True/False

glob module:

This module is used to return file/pathnames matching a specified pattern. The pattern rules of glob follow standard unix path expansion rules.

Glob( ):

```
import glob
for name in glob.glob('*.txt');
for name in glob.glob('*');
for name in glob.glob('?.txt');
for name in glob.glob('*[0-9]*');
```

Re Module: Regular Expression:

```
import re
txt = "Hai Hello"
x = re.search("Hai", txt)
x = re.sub( )
x = re.split()
x = re.sub( ),
```

Python - Os Module

It is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

Creating Directory :-

We can create a new directory using the mkdir( ) function from the Os module.

```
>>> import os
>>> os.mkdir ("d:\\tempdir")
```

A new directory corresponding to the path in the string argument of the function will be created. If we open D drive in Windows Explorer, we should notice tempdir folder created.

Changing the Current Working Directory

We must first change the current working directory to a newly created one before doing any operations in it. This is done using the Chdir ( ) function.

```
>>> import os
>>> os.chdir("d:\\tempdir")
```

There is a getcwd() function in the os module using which we can confirm if the Current working directory has been changed or not.

```
>>> os.getcwd()
'd:\\tempdir'
```

Directory paths can also be relative. If the Current directory is set to D drive and then to tempdir without mentioning the Preceding path, then also the current working directory will be changed to d:\tempdir.

```
>>> os.chdir("d:\\")
>>> os.getcwd()
'd:\\'
>>> os.chdir("tempdir")
>>> os.getcwd()
'd:\\tempdir'
```

In order to set the current directory to the parent directory use ".." as the argument in the chdir() function.

```
>>>os.chdir("d:\\temdir")
>>>os.getcwd()
'd:\\tempdir'
>>>os.chdir("..")
>>>os.getcwd()
'd:\\'
```

<u>Removing a Directory</u>:— The rmdir() function in the OS module removes the specified directory either with an absolute or relative Path. However, we can not remove the Current working directory. Also, for a directory to be removed, it should be empty. For example, tempdir will not be removed if it is the current directory. We have to change the Current working directory and then remove tempdir.

```
>>>os.chdir("tempdir")
>>>os.getcwd()
'd:\\tempdir'
>>>os.rmdir("d:\\tempdir")
PermissionError: [WinError 32] The process
Cannot access the file be-cause it is being
Used by another process: 'd:\\tempdir'
>>>os.chdir("..")
>>>os.rmdir("tempdir")
```

List files and Sub-directories

The listdir() function returns the list of all files and directories in the specified directory.

```
>>> os.listdir("C:\python37")
['DLLS', 'DOC', 'fantasy-1.Py', 'fantasy.db',
'fantasy.Py', 'frame.py', 'gridexample.py',
'include', 'Lib', 'Libs', 'LICENSE.txt',
'listbox.py', 'NEWS.txt', 'place.Py',
'Players.db', 'Python.exe', 'python3.dll',
'Python36.dll', 'Pythonw.exe', 'Sclst.Py',
'Scripts', 'tcl', 'test.py', 'Tools', 'tooltip.
Py', 'Vcruntime140.dll', 'Virat.jpg',
'Virat.Py']
```

If we don't specify any directory, then list of files and directories in the current working directory will be returned.

```
>>> os.listdir()
[':Config', '.dotnet', 'python']
```

Python - Sys Module:- The Sys module Provides functions and variables used to manipulate different parts of the python runtime environment. You will learn Some of the important features of this module here.

Sys.argv

Returns a list of command line arguments paassed to a python script. The item at index 0 in this list is always the name of the script. The rest of the arguments are stored at the subsequent indices.

Here is a python script (test.py) Consuming two arguments from the command line

Example: test.py

```
import sys
Print ("Hello {.} .Welcome to {} tutorial".
format (Sys.argv[1], Sys.argv[2]))
```

This script is executed from command line as follows:

C:\Python 36 > Python test . Py Matin Python
Hello Martin . Welcome to python tutorial.

As you can see, sys.argv[1] contains the first parameter 'Martin', while sys.argv [2] Contains the second parameter 'python'. Sys . argv[0] Contains the Script file name test.Py .

Sys.exit

This causes the Script to exit back to either the python console or the Command prompt. This is generally used to Safely exit from the program in Case of generation of an exception.

Sys. maxsize

Returns the largest integer a variable Can take.

```
>>> import sys
>>> Sys. maxsize
9223372036854775807
```

Sys.path

This is an environment variable that is a search path for all python modules.

```
>>> sys.path
['', 'C:\\Python36\\Lib\\idlelib', 'C:\\Python
36\\Python36.zip', 'C:\\Python36\\DLLS',
'C:\\Python36\\lib', 'C:\\Python36',
'C:\\users\\acer\\Appdata\\Roaming\\
Python\\Python36\\Site-Packages',
'C:\\Python36\\Lib\\Site-Packages']
```

Sys.Version

This attribute displays a string containing the version number of the current Python interpreter.

```
>>> sys.version
'3.7.0 (v3.7.0:f59c0932b4, Mar 28
2018, 17:00:18) [MSC v.1900 64 bit(AMD64)]
```

Request modules: import requests

make a request to a web page, and print the response text.

Ex: import requests

```
x = requests.get('http://www.htm')
print(x.text)
```

File handling :

Reading and writing Binary Data in python.

- Read the entire file as a Single byte String.

```
with open('filebinary.bin', 'rb') as fl:
    allBinData = fl.read()
```

- write binary data to a textfile.

```
with open('filebinary.bin', 'wb') as fl:
    fl.write(b'abc ~~Hello world~~')
```

A B C
65 66 67

65 - 1000001
66 - 1000010
67 - 1000011

97,
98,
99.

Binary file :-

	01	02	03	04
00				
01				
10				
11				

Q. parsing the Text file in python?

parsing - Analyze into logical Syntax Components.
Convert data in a certain format

Consider:

```
a b c
1 2 3
4 5 6
7 8 9
```

```
import pandas as pd
df = pd.read_csv('data.txt')
df
```

o/p

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

:- Unit - 3 :-

3.1 Numpy Arrays :

1. Numpy is a "New".
2. Numpy is a library used for arrays
3. Its also used in the domain of linear, algebra, fourier transform and matrices.
4. Numpy Created in 2005.

Use of Numpy :-

1. we have list for server purpose its fast.
2. Numpy provides an array obj upto 50x faster than, traditional python lists.
3. The array object is Numpy is called as a "ndarray" its make Easy to done work.

array

Why Numpy are fast :

Numpy | a | b | c |

List | a | c | b |

```
class Numpy //class
{
    void array() //meth
    {
    }
}
class Main
{
    main()
    {
    obj //ndarray = Numpy()
    }
}

ndarray.array();
```

Install Numpy :

pip install numpy

Import Numpy.

import numpy.

Ex: import numpy

```
arr = numpy.array([1,2,3])
print(arr).
```

o/p : [1, 2, 3]

Numpy Version.

```
import numpy as n
print(n.__version__)
```

o/p : 1.16.3

3.2 Creating Arrays using Numpy :

The array object is called ndarray.

we create ndarray object with array()

Code :

```
import numpy as np
a = np.array([1,2,3])
print(a)
print(type(a)).
```

Dimensions of Arrays :

0-D
1 D
2 D
3 D

1. 0 Dimension :

```
import numpy as np
a = np.array(10)
print(a)
```

1 Dimension:

```
import numpy as np
a = np.array([1,2,3])
print(a)
```

2 Dimensional:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a).
```

3 - Dimensional:

```
import numpy as np
a = np.array([[[1,2,3],[1,2,3]],[[1,2,3],[1,2,3]]])
print(a).
```

Checking "ndim"

```
import numpy as a
s = a array([1,2,3])
print(s.ndim)
```

[[[ ],[ ]],
[[ ],[ ]]]

3.3 Array Indexing: ✓

notly but [10] a[0]

Slicing Array:

we pass slice instead of index [start : End]

[Start : End : Step]

Start = 0

End = Length of Dimension

Step = 1

Ex:

```
import numpy as n
a = n.array([1,2,3], [1,2,3])
print( a[1][2]) = 3.
```

~~print(.we can slice on only 8ing[] array.~~

```
print( a[1, 1:4])
```

3.4. Fancy Indexing:

In this we pass an a array to the indices to access multiple array elements at once.

Code:

```
import numpy as np
rand = np.random.RandomState(42)
x = rand.randint(100, size = 10)
print(x)
```

o/p: [10, 11 21 35 10 65 72 89 90 19]

```
[x[3], x[7], x[2]]
```

o/p: 35 89 21

```
a = [3,7,4]
x[a]
```

op: [35, 89 10]

```
a = np.array([[3,7],
              [4,5]])
x[a]
```

op: ([[35, 89],
[10, 65]])

3.6 Numerical Operations on Numpy Array

We have the numerical operations that are the add, sub, mul, div,

```
import numpy as np
a = np.arange(9).reshape(3,3)
print("Array A")
print(a)
b = np.array([10,10,10])
print("Array B")
print(b)
print("Addition"),
print(np.add(a,b))
print("Subtraction")
print(np.subtract(a,b))
print("Multiplication")
print(np.multiply(a,b))
print("Division");
print(np.divide(a,b)).
```

A = ([0 1 2
3 4 5
6 7 8]]

B = [10, 10, 10]

Output:

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

Array B
[10, 10, 10]
Addition
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

Subtraction

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

Multiply:

```
[[0   10  20]
 [30  40  50]
 [60  70  80]]
```

divide:

```
[[0.   0.1  0.2]
 [0.3  0.4  0.5]
 [0.6  0.7  0.8]]
```

3.6 Array Functions:

Arrays are used to store multiple values in one single variable. python doest not have built-in Support for Arrays, but python lists can be used instead and have the functions.

Functions are:

1. zeros( ) ✓
2. ones( ) ✓
3. reshape( )
4. arange( )
5. linespace( )
6. zeros-like( ) ✓
7. once-like( ) ✓
8. full-like( )
9. Eye( )
10. Empty( ) ✓
11. Empty-like( ) ✓
12. identity( )

1. Zeros ( );

Return a new array of given shape & type filled with zeros.

Syntax:

numpy.zeros(shape, dtype = None, order = 'c')

Ex: a = numpy.zeros([2, 2], dtype = int)

[[0,0],
[0,0]]

2. ones( ):

Return all make as ones.

Syntax:

numpy.ones(shape, dtype = None, order = 'c')

Ex:

a = numpy.ones([2, 2], dtype = int)

[[1,1],
[1,1]]

3. ones_like( );

- an array of given shape and type as given array with ones.

Syntax:

numpy.ones_like(array, dtype = None, order = 'c')

Ex: numpy.ones_like(array)

[1, 1, 1, 1, 1 1 1 1]

4. zeros_like();

- all the values to the zeros.

Syntax:

numpy.zeros_like(array, dtype, order='c');

Ex:

numpy.zeros_like(array)

[0 0 0 0 0 0 0 0]

5. reshape();

- Shapes an array without changing data of array.

Syntax:

numpy.reshape(array, shape, order='C')

Ex: ~~a = numpy.reshape~~

a = numpy.arange(8).reshape(2,4);

[[0 1 2 3]
[4 5 6 7]]

6. arange();

It can make that to range display

Syntax:

numpy.arange((Start), (Stop))

Ex: a = numpy.arange(8).

(2, 4)

7. linespace();

returns evenly spaced values within a given interval. num no. of elements are returned.

Syntax:

numpy.linespace(start, stop, num = 50, Endpoint = True, retstep = False, dtype = None);

Ex:

a = numpy.linespace(2.0, 3.0, num = 5, retstep = True);

op:

array([2. , 2.25, 2.5 , 2.75, 3.]), 0.25)

8. full-like();

returns a new array with the same shape and type of a given array.

Syntax:

numpy.full-like(a, fill-value, dtype = None, order 'K', subok = True)

Ex:

a = numpy.full-like(a, 0.01)

$$\begin{bmatrix} 0.01 & 0.01 & 0.01 \\ 0.01 & 0.01 & 0.01 \end{bmatrix}$$

9. Eye();

- a 2D array with ones on the diagonals and zero elsewhere.

Syntax:

numpy.Eye(R, C=None, K=0, dtype='float')

Ex:

a = numpy.Eye(2, dtype = float)

op:

```
[[1. 0.]
 [0. 1.]]
```

10. Empty();

returns a new array without initializing entries.

Syntax:

numpy.Empty(shape, dtype=float, order='C')

Ex: a = numpy.Empty([2, 2], dtype=int)

```
[[0 0]
 [0 0]]
```

11. Empty_likes();

new array as a given array.

Syntax:

numpy.empty-like(a, dtype=None);

Ex: a = numpy.Empty-like([2, 2], dtype=int)

```
[[0 0]
 [0 0]]
```

12. identity(); - return an identity array.

Syntax: numpy.identity(n, dtype=none)

Ex: a = numpy.identity(2) // [[1 0] [0 1]]

3.7 Data processing using Arrays:

With the Numpy package, we can easily solve many kinds of data processing tasks without writing complex loops. It is very helpful for us to control our code as well as the performance of the program. In this part, we want to introduce some mathematical and statistical functions:

Functions:

a. sum( )
b. prod( )
c. diff( )
d. gradient( )
e. cross( )

a. Sum( ):

Calculate the sum of all the elements in an array or along the axise.

Ex:

```
a = np.array([[2,4],[3,5]])
np.sum(a, axis=0)
array([5,9])
```

b. prod( ):

0+1 = 1

Compute the product of array elements over the given axis.

Ex:

```
np.prod(a, axis=1)
array([8,15])
```

0 * 1 = 0

c. diff();

Calculate the discrete difference along the given axis.

Ex:

```
np.diff(a, axis=0)
array([[1, 1]])
```

d. gradient();

return the gradient of an array.

Ex: np.gradient(a)

```
[array([[1., 1.], [1., 1.]]),
 array([[2., 2.], [2., 2.]])]
```

e. Cross();

Returns the cross product of two (arrays of) vectors.

Ex:

```
np.cross(x, y).
x = [1, 2, 3]
y = [4, 5, 6]
np.cross(x, y)
array([-3, 6, -3])
```

3.8 Loading and Saving data in array.

Loading data array: numpy.loadtxt( )

The process of load the data from the text file to the numpy library. for ~~Syntax~~: faster reading.

Syntax:

```
numpy.loadtxt( fname, dtype='float',
               comments='#', delimiter=None,
               converters=None, skiprows=0,
               usecols=None, unpack=False,
               ndmin=0)
```

Saving data array:

Numpy.savetxt( ):

is a method in python in numpy library to save an 1D and 2D array to a file.

Syntax:

```
numpy.savetxt(fname, x, fmt='%.18e',
              delimiter='', newline='\n',
              header='', footer='', comments='#',
              encoding=None).
```

Saving and loading 3D arrays.

1. reshape the 3D array to 2D array.
2. Insert this array to the file.
3. Load data from the file to display.
4. Convert back to the original shapped array.

3.9. Loading an Array:

numpy.load() function retcan the input array from a disk file with npy extension. (.npy).

Syntax:

```
numpy.load(file, mmap_mode = None,
           allow_pickle = True, fix_imports = True,
           encoding = 'ASCII')
```

Ex:

```
import numpy as np
a = np.array([[i+j for j in range(3)
               for j in range(3)]])
print(a)
np.save('myload', a)
print("Saved in my load.npy").
b = np.Load('myload.npy')
print(b)
```

output:

a is:

[0, 1, 2, 1, 2, 3, 2, 3, 4]

Saved in my load.npy

b is:

[0, 1, 2, 1, 2, 3, 2, 3, 4]

// its also printed from ~~go~~ myload file.npy.

3.10 Saving an Array:

Saving is a numpy.save() function is used to store the input array in a disk file with npy extension (.npy).

Syntax:

numpy.save(file, arr, allow_pickle = True, fix_imports = True).

Ex:

write ex program as loading an array.

3.11 Linear Algebra in Numpy:

The linear Algebra module of Numpy offers various methods to apply linear algebra on any numpy array.

one can find.

1. rank, determinant, trace etc. of an array.
2. Eigen values of matrices
3. Matrix and vector products (dot, inner, outer, etc. product), matrix Exponentiation.
4. Solve linear or tensor Equations and much more!

program for rank, trace, inverse, Det

```
import numpy as np
a = np.array([ [6, 1, 1],
               [4, -2, 5],
               [2, 8, 7]])
```

```
print(" Rank of A:", np.linalg.matrix_rank(a))
print("\nTrace of A:", np.trace(a))
print("\n Determinant of A:", np.linalg.det(a))
print("\n Inverse of A", np.linalg.inv(a))
print("\n matrix power", np.linalg.matrix_power(A,3))
```

output:

```
Rank of A : 3
Trance of A : 11
Determinent of A: -306.0
Inverse of A:

[[0.011  -0.003  -0.022]
 [0.05   -0.13    0.08]
 [-0.114  0.15    0.05]]

Matrix A raised to power 3:

[[336  162  228]
 [406  162  469]
 [698  702  905]]
```

matrix eigenvalues functions:

numpy.linalg.eigh(a, UPLO='L')

- used to return the eigenvalues and eigenvectors. of Complex Hermitation (Conjugate Symetric)

Program:

```
from numpy import linalg as np
a = np.array([[1,-2i],[2i, 5]])
print(a)
c,d = np.eigh(a)  print(c)  print(d)
```

Matrix and vector products:

Dot product

```
numpy.dot(va, vb, out = None);
```

Ex:

```
import numpy as np
product = np.dot(5, 4)
print(" Dot product of Scalar Values ", product)
va = 2+3j
vb = 4+5j
product = np.dot(va, vb);
print(" Dot product : ", product)
```

op:

va = 2+3j

vb = 4+5j

now dot product

$\Rightarrow 2(4+5j) + 3j(4 + -5j)$

$\Rightarrow 8 + 10j + 12j - 15$

$\Rightarrow 8 + 22j - 15$

$\Rightarrow -7 + 22j$ //

$(2+3j) + (4+5j)$

$8 + 10j + 12 + 15(j)$

$8 + 10j + 12 - 15$

$8 + 22j - 15$

$-7 + 22j$ //

```
numpy.vdot(va, vb);
```

Solving Equation and inverting matrices:

Solve a linear matrix Equation : $ax = b$.

Program.

```
import numpy as np
a = np.array([[1,2],[3,4]])
b = np.array([8,18])
print(np.linalg.solve(a,b)))
```

output:

op: [2. 3.]

a=[[10,10],[20,10]]
np.sum(a,axis=0)

3.12 Numpy Random Numbers:

Random number does NOT mean a different number Every time. Random means something that can not be predicted logically.

Generate Random Number: (int)

```
from numpy import random
x = random.randint(5)
print(x)
```

o/p : 3

Generate Random float:

```
from numpy import random
x = random.rand( )
print(x)
```

o/p: 0.57014320.

Generate Random Array:

1D array:

```
from numpy import random
x = random.randint(100, size=(5))
print(x)
```

output:

```
[10 51 60 99 27]
```

2D array:

```
from numpy import random
x = random.randint(100, size=(2,5))
print(x)
```

```
[ [27 69 66 71 70]
  [85 75 86 75 1]]
```

Generate Random Number from Array:

```
from numpy import random
x = random.choice([3,5,7,9])
print(x)
```

o/p: 7

Random Distrubtion:

It is a set of random numbers that follow a certain probability density function.

Ex:

```
from numpy import random
x = random.choice([3,5,7,9],
                  p=[0.1,0.3,0.6,0.0], size=(3,5))
print(x).
```

output:

```
[[7 7 3 5 7]
 [7 7 5 7 7]
 [7 7 7 5 5]]
```

3.13. Pands:

pands is a python library used for working with data sets.

It has a function for analyzing, cleaning and Exploring and manuplating data.

The name "Pandas" has refers to the both "panel Data", and "python Data Analysis" and was Created by wes Mckinney in 2008.

Uses:

It allow to analyze big data and make Conclusions based on Stastical theories.

pandas can Easily messy data sets, and make the readable and relevant.

Relevant data is very important in the data sciences.

What pandas can do?

There are correlation between two or more columns.?

What is average value?

Max value?

Min value?

Importing pands:

```
import pandas
```

Syntax:

dataset(vs) = {'Table-name': ['val', 'val', ...]}

Creating a panda:

```
import pandas
myds = { 'cars' : ['aa', 'bb', 'cc'] }
print(pandas.DataFrame(myds));
```

output:

```
import pandas
myds = {'Shapes' : ['circle', 'Triangle', 'Squae']}
print(pandas.DataFrame(myds));
```

output:

```
  Shapes
0 Circle
1 Triangle
2 Squae
```

3.14. Panda Packages :

panda is a python package providing fast, flexible and Expansive data structure designed to make working with "relational" & "labelled" data both Easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in the python.

There are two primary data structures of the pandas that are.

1. Data Structures - series
2. Data Frames.

1. Data Structure / Series :

pandas series is one-dimensional labeled array capable of holding data of any type(int, Str, Float, python objects etc..

" The axies lables are Correctly called as indexes".

pandas series is nothing but a column in a excel sheet.

Labels need not be unique but must be a hashable type. The object supports both int and label-based indexing and provides a host of methods for performing operations involving the index.

ser = pd.Series(df['sname'])

ser = pd.Series(df['sid'])

	sname	sid
0	aa	101
1	bb	102
2	cc	103
3	dd	104

1. Creating a pandas Series:

panda series will be created by loading the datasets from existing storage, Storage can be SQL, DataBase, CSV files, and Excel files. Pandas Series can be created from the lists, dictionaries, and from a scalar value etc. Series can be created in different ways; here are some ways by which we create a series.

Creating a series from array :

```
import pandas as pd
import numpy as np
data = np.array(['c', 'h', 'a', 'n', 'd', 'u'])
ser = pd.Series(data)
print(ser)
```

o/p:

```
0 c
1 h
2 a
3 n
4 d
5 u
dtype: object
```

Creating a series from lists :-

```
import pandas as pd
li = ['a', 'b', 'c']
ser = pd.Series(li)
print(ser)
```

o/p:

```
0 a
1 b
2 c
dtype: object
```

Accessing the Element from Series with position.

we can be access the Element through the two ways that are the...

a: Accessing Element from Series with position.
b. Accessing Element using Label(index).

a. Accessing Element from Series with position

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d','e'])
sere = pd.Series(data)
print(sere[:5]
```

output:

```
0 a
1 b
2 c
3 d
4 e
dtype: object
```

b. Accessing Element using Label (index):

```
import pandas as pd
import numpy as np
data= np.array(['a','b','c'])
sere = pd.Series(data, index = [1,2,3])
print(sere(2))
```

Output:

b.

Indexing and Selecting Data ~~from~~ in a Series

Indexing in pandas means simply selecting particular data from a Series. Indexing could mean selecting all the data, some of the data for particular columns. Indexing can also be known as "Subset Selection".

Indexing a Series using indexing operator [ ]

Indexing operator is used to refer to the square brackets following an object. The .loc and .iloc indexers also use the indexing operator to make selections. In this indexing operator to refer to df[ ].

```
import pandas as pd
df = pd.read_csv("nba.csv")
ser = pd.Series(df['Name'])
data = ser.head(3)
data.
```

Output:

```
0 Raman
1 Chandan
2 Rajan
```

Now, use the indexing operator [ ]

```
data[1:2]

data[1:2]
```

o/p:

```
0 Raman
1 Chandan
```

Indexing using ".loc[ ]"

The df.loc indexer selects data in a different way than just the indexing operator. It can also select subsets of data.

```
impost pandas as pd
df = pd.read_csv("nba.csv")
ser = pd.Series(df['Name'])
data = ser.head(10)
data.loc[3:6]
```

both include

Output

```
3   aaa
4   bbb
5   ccc
6   ddd
Name: Name, dtype: object
```

Indexing using ".iloc[ ]"

The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

```
impost pandas as pd
df = pd.read_csv("nba.csv")
ser = pd.Series(df['Name'])
data = ser.head(10)
data.iloc[3:6]
```

Exclude

Output:

```
3   aaa
4   bbb
5   ccc
```

b. Data Frames:

A Data Frame is a two-dimensional data structure, i.e data is aligned in a tabular fashion in rows and the columns.

Features:

1. potentially Columns of different types
2. Size-Mutable
3. Labeled axes (r & c)
4. Can perform Arithmetic operations on row & Colu

Structure:

	C_1	C_2
	Sid	Sname
$\rightarrow R_1$	101	aaa
$\rightarrow R_2$	102	bbb
$\rightarrow R_3$	103	ccc

pandas. DataFrame:

A pandas DataFrame can be created using the following Constructor.

Syntax:

pandas.DataFrame(data, index, columns, dtype, copy)

Create a Data Frame:

A Data Frame can be created using various inputs like

1. lists
2. dict
3. Series
4. Numpy ndarrays
5. Another Data Frame.

Create an Empty Data Frame:

```
import pandas as pd
df = pd.Dataframe()
print(df)
```

o/p

```
Empty DataFrame
Columns: []
Index: []
```

Create a Data frame with Lists.

Ex:1

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print(df)
```

o/p

	0
0	1
1	2
2	3
3	4
4	5

Ex:2

```
import pandas as pd
data = [["aa", 101], ["bb", 102], ["cc", 103]]
df = pd.DataFrame(data, columns = ['Name', 'Age'], dtype = float)
print(df)
```

output:

	Name	Age
0	aa	101.0
1	bb	102.0
2	cc	103.0

Create a DataFrame from Dict of ndarrays/lists

All the ndarrays must be of same length. If index is passed, then the length of the index should equal to length of arrays.

```
import pandas as pd
data = {'Name': ['aa', 'bb', 'cc'],
        'Age': [10, 12, 13]}
df = pd.DataFrame(data, index = ['r1', 'r2', 'r3'])
print(df)
```

output:

	Age	Name
r1	10	aa
r2	12	bb
r3	13	cc

Create a Dataframe from List of Dicts:

Example:1)

```
import pandas as pd
data = [{'a':1, 'b':2}, {'a':5, 'c':20}]
df = pd.DataFrame(data, index=['First', 'Second'])
print(df)
```

optional

output:

	a	b	c
First	1	2	NaN
Second	5	10	20.0

Create a Dataframe from Dict to Series

```
import pandas as pd
df = {'one': pd.Series([1,2,3], index=['a','b','c']),
      'two': pd.Series([1,2,3,4], index=['a','b','c','d'])}
df = pd.DataFrame(d)
print(df)
```

Output:

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Column, addition and deletion through Examples

Column Selection:

```
import pandas as pd
d = {'one': pd.Series([1,2,3]), index=['a','b','c']),
     'two': pd.Series([1,2,3,4]), index={'a','b','c','d'}}
df = pd.DataFrame(d)
print(df['one'])
```

output:

a	1.0
b	2.0
c	3.0
d	NaN

Column Addition:

```
import pandas as pd
d = {'one': pd.Series([1,2,3], index=['a','b','c']),
     'two': pd.Series([1,2,3,4], index=['a','b','c','d'])}
df = pd.DataFrame(d)

#adding a new column.
print("Adding a new colum by passing as Series:")
df['three'] = pd.Series([10,20,30], index=['a','b','c'])
print(df)

print("Adding new Column using the Existing Columns in
      DataFrame:")
df['four'] = df['one'] + df['three']
print(df).
```

Output:

Adding a new Colu by passing Series:

	one	two	three
a	1.0	1	10.0
b	2.0	2	20.0
c	3.0	3	30.0
d	NaN	4	NaN

Adding new Column using the Existance Colu in DF

	one	two	three	four
a	1.0	1	10.0	11.0
b	2.0	2	20.0	22.0
c	3.0	3	30.0	33.0
d	NaN	4	NaN	Nan.

Column Deletion:

```
import pandas as pd
d = {'one': pd.Series([1,2,3], index=['a','b','c']),
     'two': pd.Series([1,2,3,4], index=['a','b','c','d']),
     'three': pd.Series([10,20,30], index=['a','b','c'])}
df = pd.DataFrame(d)
print("our dataframe is:")
print(df)

# using del fun.
del df['one']
print(df)

# using pop function
df.pop('two')
print df
```

Output:

```
our dataframe is
   one  three  two
a  1.0   10.0    1
b  2.0   20.0    2
c  3.0   30.0    3
d  NaN    NaN    4
// Delete //
   three  two
a   10.0    1
b   20.0    2
c   30.0    3
d    NaN    4
// pop
   three
a   10.0
b   20.0
c   30.0
d    NaN
```

Row Selection, Addition, Deletion:

Row Selection:

passing row label to a loc function.

```
import pandas as pd
d = {'one' : pd.Series([1,2,3], index=['a','b','c']),
     'two' : pd.Series([1,2,3,4], index=['a','b','c','d'])}
df = pd.DataFrame(d)
print(df.loc['b'])
```

Output:

```
one    2.0
two    2.0
~~three~~ Name : b, dtype : float 64,
```

Selection by integer rows:

```
print(df.loc[2])
```

O/P:

```
one    2.0
two    2.0
Name : b, dtype : float 64.
```

Slice Rows:

```
print(df[2:4])
```

output:

	one	two
c	3.0	3
d	NaN	4

Addition of Rows:

```
import pandas as pd
d1 = pd.DataFrame([[1,2],[3,4]], columns=['a','b'])
d2 = pd.DataFrame([[4,5],[6,7]], columns=['a','b'])
d3 = d1.append(d2)
print(d3)
```

output

	a	b
0	1	2
1	3	4
0	4	5
1	6	7

Deleting of Rows:

```
d3=dB.drop(0)
print(df)
```

42 Output:

67

	a	b
1	3	4
1	6	7

42 - 52 62 67
10 10
+5
26

The Essential basic functionalies:

1. Reindexing
2. Altering lables.
3. heads & tails in pandas.
4. Functional Stestics in pandas.
5. Functional Sorting in pandas
6. Functional Indexing in pandas.
7. Functional Selecting Data.
8. pandas Computational tools.
9. working with missing data.

3.15 1. Reindexing in pythons:

Reindexing changes the row labels and Column labels of a DataFrame. To reindex means to Conform the data to match a given set of labels along a particular axis.

Multiple operations can be accomplished through indexing like.

a. Reorder the existing data to match a new set of lab
b Insert missing value (NA) marker in label locations where no data for the label existed.

```
import pandas as pd
import numpy as np. //
```

we can make that row reindexing and the column reindexing that the data.

Reindexing the Rows:

one can reindex a signal row or multiple rows by using the "reindex()" method. Default values in the new index that are not present in the dataframe are assigned. NaN.

Example:

```
import pandas as pd
import numpy as p
Column = ['a', 'b', 'c', 'd', 'e']
index = ['A', 'B', 'C', 'D', 'E']
df1 = pd. DataFrame (np. random. rand (5, 5),
                     Columns = Column, index = index)
print(df1)
df1. reindex ( ['B', 'D', 'A', 'C', 'E'])
```

output:

	a	b	c	d	e
A	0.10	0.60	0.08	0.03	0.13
B	0.20	0.70	0.07	0.02	0.14
C	0.30	0.80	0.06	0.01	0.15
D	0.40	0.90	0.05	0.11	0.16
E	0.50	0.09	0.04	0.12	0.17

Dataframe after reindexing row:

	a	b	c	d	e
B	0.20	0.70	0.07	0.02	0.14
D	0.40	0.90	0.05	0.11	0.16
A	0.10	0.60	0.08	0.03	0.13
C	0.30	0.80	0.06	0.01	0.18
E	0.50	0.09	0.04	0.12	0.17

Reindexing the columns using "axis" keyword.

```
import pandas as pd
import numpy as np
Column = ['a', 'b', 'c', 'd', 'e']
index = ['A', 'B', 'C', 'D', 'E']
df1 = pd.Dataframe(np.random.rand(5,5),
                   Columns = Column, index = index)
Colum = ['e', 'a', 'b', 'c', 'd']
print(df1.reindex(Colum, axis = 'Columns'))
```

Output:

	e	a	b	c	d
A	0.14	0.20	0.70	0.07	0.02
B	0.16	0.40	0.90	0.08	0.11
C	0.13	0.10	0.60	0.08	0.03
D	0.25	0.30	0.80	0.06	0.01
E	0.17	0.50	0.09	0.04	0.12

Replacing the missing values : $\boxed{\text{fill-value}}$

```
import pandas as pd
import numpy as np
Column = [ 'a', 'B', 'c', 'd', 'e']
index = [ 'A', 'B', 'C', 'D', 'E']
df1 = pd.Dataframe(np.random.rand(5,5),
                   Columns = Column, index = index)
Colum = ['a', 'b', 'c', 'g', 'h']
print(df1.reindex(Colum, axis = 'Columns', fill_value = 1.5))
```

Output:

	a	b	c	g	h
A	0.14	0.20	0.70	1.5	1.5
B	0.16	0.40	0.90	1.5	1.5
C	0.13	0.10	0.60	1.5	1.5
D	0.25	0.30	0.80	1.5	1.5
E	0.17	0.50	0.09	1.5	1.5

3.16 2. Altering Tables :

Alter means is used to add, delete/drop or modify columns in the existing table.

Create a Dataframe using dictionary.

```
import pandas as pd
df = pd.Dataframe( {"Name": ['aa', 'bb', 'cc'],
                    "Age": [ 15, 16, 19] })
print(df)
```

Output:

	Name	Age
0	aa	15
1	bb	16
2	cc	19

Changing the Column name and row indexing using df.columns and df.index attribute

```
df.columns = ['Col1', 'Col2']
df.index = ['Ra1', 'Ra2', 'Ra3']
df
```

output:

	Col1	Col2
Ra1	aa	15
Ra2	bb	16
Ra3	cc	19

```
df = df.rename(columns = {"Col1": "Mod_Col"})
```

using values attribute to rename the Colums

```
df.columns.values[0] = 'FName'
```

Befor:

	Name	Age
0	aa	10
1	bb	20
2	cc	30

After

	FName	Age
0	aa	10
1	bb	20
2	cc	30.

Change row index by Lambda function.

```
df = pd.DataFrame({"A": ['aa', 'bb', 'cc'],
                   "B": [10, 20, 30]})
df = df.rename(index = lambda x: x+10)
df
```

	A	B
10	aa	10
11	bb	20
12	cc	30

3.10 3. Head and tails in pandas:

Let's discuss how to select top or bottom N number of rows from a Dataframe using the head() & tail() methods.

1. Select first N Rows from a Dataframe using head() method of pandas Dataframes.

pandas head() method is used to return top n (5 by default) rows of a Data frame & series

Syntax:

$\boxed{\text{Dataframe.head(n)}}$

- n is an integer value, no. of rows to be retrieved.
- return data frame with top n rows.

Create a Dataframe:

```
import pands as pd
Students = [('Ram', 10),
            ('Syam', 10),
            ('Shyas', 10), ('Raj', 10), ('Lalu', 10)]
det = pd.DataFrame(Students, Columns=['Name',
                                      'Score'],
                   index = ['a', 'b', 'c', 'd', 'e'])
det
```

Output:

	Name	Score
a	~~Ram~~	10
b	~~bb~~	10
c	~~cc~~	10

	Name	Score
a	Ram	10
b	Syam	10
c	Shyas	10
d	Raj	10
e	Lalu	10

To display 1st five Rows:

```
df.head()
```

	Name	Score
a	Ram	10
b	Syam	10
c	Sreys	10
d	Raj	10
e	Kal.	10

Show top 1 row

```
df[['Score']].head(1)
```

o/p:

10

To display only three:

```
df.head(3)
```

	Name	Score
a	Ram	10
b	Syam	10
c	Sreys	10

Selecting has N. Rows a from DataFrame using (tail) method in pandas.

By default 5 rows

Syntax:

```
DataFrame.tail(n)
```

Example:

```
import pandas as np
stu = [('aa', 10),
       ('bb', 20),
       ('cc', 30),
       ('dd', 40),
       ('ee', 50)]
df = np.DataFrame(stu, columns=('name', 'score'))
df
```

	Name	Score
0	aa	10
1	bb	20
2	cc	30
3	dd	40
4.	ee	50

display last five Rows

df.tail()

	Name	Score
0	aa	10
1	bb	20
2	cc	30
3	dd	40
4	ee	50

To display only three:

df.tail(3)

	Name	Score
2	cc	30
3	dd	40
4	ee	50

Show bottom row '2'

df[['Name', 'Age']].tail(2)

O/P:-

	Name	Age
1	dd	40
2	ee	50

3:18

4. Functional Statistics in pandas:

Statistical methods help in the understanding and analyzing the behavior of data. We will now learn a few statistical functions we can apply on the python objects.

Statistics Functions:

1. pct_change()
2. cov() — Covariance
3. corr() — Correlation
4. rank()
   - → avg()
   - → min()
   - → max()
   - → first()

1. pct_change():

Series, DataFrame and panel, all have the function [pct_change()]. "Compares every element with its prior element and computes that change percentage.

```
import numpy as np
import pandas as pd
s = pd.Series([1, 2, 3, 4, 5, 4])
print(s.pct_change())
df = pd.DataFrame(np.random.randn(5,2))
print(df.pct_change())
```

Output:

```
0     NaN
1    1.000000
2    0.500000
3    0.333333
4    0.250000
5   -0.200000
dtype: float64
```

	0	1
0	NaN	NaN
1	-15.151902	0.174730
2	-0.746374	-1.449010
3	-3.582229	-3.165836
4	15.601180	-1.860434

2. Convariance: (Same direction +ve & -ve)

It is applied on serial data. The serial object has a method (cov) to compute convariance b/w serial objects. NA will be excluded automatically.

```
import pandas as pd
import numpy as np
s1 = pd.Series(np.random.randn(10))
s2 = pd.Series(np.random.randn(10))
```

output:

-0.129780053324.

3. Correlation: it varies b/w -1 & +1.

It shows the linear relationship between any two arrays of values (series). There are multiple methods to compute the correlation like pearson (default), Spearman and kendall.

```
import pandas as pd
import numpy as np
frame = pd.DataFrame(np.random.randn(10,5),
                     Columns = ['a', 'b', 'c', 'd', 'e'])
print(frame['a'].corr(frame['b']))
print(frame.corr())
```

Output:

-0.383712785514

	a	b	c	d	e
a	1.00..	-0....	-0.	0.	-0.
b	-0....	1....	0.	-0.	0.
c	-0....	0....	1.	-0.	0.
d	-0....	-0....	-0.	1.	0.
e	-0....	0....	-0.	-0.	1.

3.17 Data Ranking :-

It assign the rank to all the Elements if that are two are "tie" then allote mean rank

```
import pandas as pd
import numpy as np
s = pd.Series(np.random.np.random.randn(5),
              index=list('abcde'))
s['d'] = s['b']  # here tie.
print(s.rank())
```

output:

```
a  1.0
b  3.5
c  2.0
d  3.5
e  5.0
dtype: float64
```

Consider a table:

Sname	Sage	Smarks
aaa	10	100
bbb	12	98
ccc	15	95
ddd	14	75
eee	19	86

→ 'stu.csv'

```
impost pandas as pd
impost numpy as np
ds= pd.read_csv('sto.csv')
ds.head
```

Sname	Sage	Smarks
aaa	10	100
bbb	12	98
ccc	15	95
ddd	17	75
eee	19	86

a. mean :

```
mean = ds['Smarks'].mean()
```

- 90.8

b. median :

```
median = ds['Smarks'].median()
```

- 95

c. Mode :

```
mode = ds['Smarks'].mode()
```

- 100, 98, 95, 75, 86

d. Count :

```
count = ds['Sage'].count()
```

- 5

e. Std - Standard Deviation :

```
std = ds['Smarks'].std()
```

→ 9.23904786

f. max :

```
smax = ds['Smarks'].max()
```

- 100

g. min :

```
min = ds['Smarks'].min()
```

- 75

5. Functional Sorting in pands:

pandas sort_values() function sorts a data frame in Ascending order / Descending order of passed Columns. It's different than the sorted python function since it cannot sort a date frame and particular Column cannot be selected.

Syntax:

```
Dataframe.sort_values()
```

Sorting by the Name:

Before Sorting:

```
import pandas as pd
data = pd.read_csv("nba.csv")
data.
```

After Sorting:

```
import pandas as pd
data = pd.read_csv("nba.csv")
data.sort_values("Name", axis=0, ascending=True,
                 inplace=True, na_position='last')
data
```

Output:

Sname	Sid
aa	101
ab	102
ac	103
bb	104
bc	105
bd	106
cc	107
ca	108
ca	109

6. Indexing and Selecting Data with pandas

Indexing in pandas: [ ] "refer previous"

selecting a Single Columns:

```
import pandas as pd
data = pd.read_csv("nba.csv", index_col="Name")
first = data["Age"]
first
```

Age
10
20
30

selecting multiple Columns:

```
first = data[["Age", "name"]]
fist
```

Age	name
10	aa
20	bb
30	cc

selecting two rows three column

```
Dataframe.loc[["r1", "r2"], ["c1", "c2", "c3"]]
```

Sid	Sname	Sage
101	aa	10
102	bb	15
103	cc	19

```
Dataframe.loc[["101", "102"], ["Sid", "Sname", "Sage"]]
```

3.20 8. pandas Computational Tools:

1. Introduction to Data Structures
2. Essential basic functionality.
3. IO tools (text, csv, HDFS,)
4. (Indexing and selecting Data)
5. multiindex / advanced indexing
6. Merge, Join
7. Concat & Compare
8. Reshaping and pivot tables.

The pandas Computational tools are...

a. Convariance
b. Correlation
c. Range,
   - avg
   - min
   - max
   - Count
   - range

3.21 Working with missing Data in pandas:

Missing data can occur no information is provided for one & more items or for a whole unit. Missing Data is a very big problem in real life scenario. Missing Data can also refer to as NA values in Pandas.. In Data frame Sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed.

In pandas missing data is represented two values.

None :

NaN: NaN (an acronym for NOT a number)

pandas treats "None" and "NaN" as Essentially interchangable for indicating missing or null values. To facilitates this Convention, there are several useful functions for detecting, removing and a replacing null values in pandas.

pandas treat None and NaN as Essentially interchangable for indicating ~~re~~ missing or null values. To facilitate this Convention, there are several useful functions for detecting, removing and replacing null values in pandas. DataFrame.

1. isnull( )
2. notnull( )
3. dropna( )
4. fillna( )
5. interpolate( )

1. isnull( ):

In order to check ~~not~~ null values in pandas DataFrame, we use isnull() function this function returns data frame of Boolean values which are True for NaN values.

```
import pandas as pd
import numpy as np
dict = {'fscore': [100, 90, np.nan, 95],
        'sscore': [80, 45, 56, np.nan],
        'tscore': [np.nan, 40, 80, 98]}
df = pd.DataFrame(dict)
df.isnull( )
```

output:

	fscore	sscore	tscore
0	F	F	T
1	F	F	F
2	T	F	F
3	F	T	F

2. notnull():

In order to check null values in pandas Dataframe, we use 'notnull()' function this function return dataframe of Boolean values which are false fo NaN values:

```
import pandas as pd
import numpy as np
dict = {'aa': [10, np.nan, 20],
        'bb': [20, 30, np.nan],
        'cc': [20, np.nan, np.nan]}
df = pd.DataFrame(dict)
df.notnull()
```

output:

	aa	bb	cc
0	T	T	T
1	F	T	F
2	T	F	F

3. fillna(), replace() and interpolate().

In order to fill null values in a datasets, we use fillna(), replace() and interpolate() functions these functions replace NaN values with some values of their own. All these functions help in filling a null values in datasets or a Dataframe. Interpolate() function is basically used to fill NA values in the dataframe but it uses various interpolation technique to fill the missing values rather than hard-coding the values.

Filling the values with a Single Value :0.

```
impoot pandas as pd
impoot numpy as np
dict = {'aa' : [10, 20, np.nan],
        'bb' : [10, np.nan, 20],
        'cc' : [10, np.nan, np.nan]}
df = pd.DataFrame(dict)
df.fillna(0)
```

output:

	aa	bb	cc
0	10	10	10
1	20	0	0
2	0	20	0

values fill with previous ones:

```
df.fillna(method = 'pad')
```

pad

	aa	bb	cc
0	10	10	10
1	20	10	10
2	20	20	10.

Fill values with next ones:

```
df.fillna(method = 'bfill')
```

	aa	bb	cc
0	10	10	10
1	20	20	Nan
2	NaN	20	Nan.

* Filling null values using replace():

```
import pandas as pd
import numpy as np
data = pd.read_csv("Emp.csv")
data[10:16]
```

Op:

Sid	SName	Sadd
10	aa	Nan
11	bb	gdv
12	cc	Nan
13	~~dd~~ Nan	Nan
14	ee	vij
15	ff	Nan

```
data.replace(to_replace = np.nan, value = -99)
```

O/P:

Sid	Sname	Sadd
10	aa	-99
11	bb	gdv
12	cc	-99
13	-99	-99
14	ee	vij
15	ff	-99

using interpolate() function:

```
import pandas as pd
df = pd.DataFrame({"A": [10, 4, 5, None, 1],
                   "B": [None, 2, 54, 3, None],
                   "C": [20, 16, None, 3, 8],
                   "D": [14, 3, None, None, 6]})
```

df.

	A	B	C	D	E
0	10	4	~~NS~~	Nan	1
1	NaN	X			
2	2				
3	5				
4					

	A	B	C	D	E
0	10	nan	20	14	
1	4	2	16	3	
2	5	54	nan	nan	
3	NaN	3	3	nan	
4	1	NaN	8	6	

Drop missing values using dropna()

```
impoot pandas as pd
impoot numpy as np
dict = {'aa':[10, np.nan, 20],
        'bb':[np.nan, 10, 20],
        'cc':[10, 20, np.nan]}
df= pd.DataFrame(dict)
df.dropna()
```

op:

	aa	bb	cc
2	10	20	30.

5th Panal Data in pandas:

A panal is a 3D Container of Data. The term panel data is derived from Econometrics and is partially reasonable for the name pandas - "pan(el)-da(ta)-s."

There are the three (3) axes are these. ✓

1. item: axis 0, each item corresponds to DF Contain inside
2. major_axis: axis 1, it is index(rows) of Each of the DF
3. minor_axis: axis 2, it is Columns of Each of the DF.

pandas.panel()

It has the Constructor is:

```
pandas.panel(data, items, major_axis, minor_axis,
             dtype, copy)
```

Create panel:

A panel can be created using multiple ways like

a. From ndarrays
b. From dict of Dataframes.

From 3D ndarray:

```
import pandas as pd
import numpy as np
data = np.random.rand(2,4,5)
p = pd.Panel(data)
print(p)
```

Output:

Dimensions : 2 (items) x 4 (major-axis) x 3 (minor-axis)
Item axis : Item1 to Item2
Major axis axis : 0 to 3
Minor-axis axis : 0 to 2

Creating a Empty panel:

```
import pandas as pd
P = pd.Panel()
print(p)
```

Output:

Dimensions : 0 (items) x 0 (major axis) x 0 (minor axis)
Item axis : None
Major axis axis : None
Minor-axis axis : None

Selecting Data from panel:

```
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4,3)),
        'Item2' : pd.DataFrame(np.random.rand(4,2))}
P = pd.panel(data)
print P['Item1']
```

Output

	0	1	2
0	0.48..	-0.12..	0.9...
1	0.41...	0.8..	0.5...
2	-2.4...	0.5..	0.2..
3	-0.4..	-0.1...	1.1...

using Major axis:

print p.major_xs(1)

	Item1	Item2
0	0.417497	0.748412
1	0.896681	-0.557322
2	0.576657	NaN

using minor axis:

print p.minor_xs(1)

	Item1	Item2
0	-0.128634	-1.047032
1	0.896681	-0.557322
2	0.571668	0.431953
3	-0.440324	1.302466

9.23. Advanced uses of pandas for Data Analysis

In this section we have consider some of advance pandas use case

Hierarchical Indexing:

Hierarchical indexing providing use with a way to work with higher dimensional data in a lower dimension by structuring the data object into mutiple date object into multiple index levels on an axis

Code:

```
ss = pd.Series(np.random.rand(8), index=
[['a','b','b','b','c','c','d','d'],[0,1,0,1,0,
1,0,1]])
ss
```

output :

```
a    0    0.721652
     1    0.297784
b    0    0.271995
     1    0.125342
c    0    0.444070
     1    0.948363
d    0    0.197565
     1    0.883776
dtype: float 64.
```

In the preceding Example, we have a Series object that has two index levels. The object can be rearranged into a DataFrame using the "unstack" function. In an inverse situation, the "stack" function can be used.

```
>>> ss.unstack( )
        0          1
a   0.549211   0.420874
b   0.051516   0.715021
c   0.503072   0.720772
d   0.373034   0.207026
```

Binary Operations of Series and Dataframes

add
sum
mul
div
mod
pow

Series:

```
impost pandas as pd
a = {'a': pd.Series([10,20]),
     'b': pd.Series([10,20])}
d = pd.DataFrame(a)
print(d)
print d.add(a)
```

```
a = pd.Series([10,10])
b = pd.Series([10,10])
```

→ a.add(b)
→ a.mul(b)
→ a.sub(b)
→ a.div(b)
→ a.mod(b)
→ a.pow(b,3)

UNIT-4

DATA Analysis Application Examples:

1. Data munging
2. cleaning Data
3. Filtering Data
4. Merging Data
5. Reshapping Data
6. Aggrigation.
7. Grouping Data:

1. Data Munging:

Data Munging is also known as a Data wrating is a key Concept of any data Science project.

It can make transforms "raw data" for making it more Suitable for analysis and it will improve the quality of your Data.

2. Cleaning Data:

Ex: isnull()

clean that to fill the Data by fillna()

```
import pandas as pd
a = {'aa': pd.Series([10,nan,30], index=[a,b,c]),
     'bb': pd.Series([nan, nan, 40], index=[a,b,c])}
b = pd.DataFrame(a)
b.fillna(1)
print(b)
```

	aa	bb
a	10	1
b	1	1
c	30	40

3. Filtering Data:

This function is used to subset rows or columns of data frame according to lables in the specified index.

Ex:

```
import pandas as pd
a = {'aa': pd.Series([1,2,3]),
     'bb': pd.Series([1,2,3]),
     'cc': pd.Series([1,2,3])}
b = pd.DataFrame(a)
b.filter(["aa","cc"])
```

O/P:

aa	cc
1	1
2	2
3	3

	aa	bb	cc
0	1	1	1
1	2	2	2
2	3	3	3

4. Merging Data:

The process of combaining is know as merging.

- concat();

```
import pandas as pd
a1 = pd.Series([1,2,3])
a2 = pd.Series([1,2,3])
b1 = pd.DataFram(a1)
b2 = pd.DataFram(a2)
pd.concat([a1,a2,b1,b2])
```

a1	a2
a1	10
a2	20

bb	bv
b1	10
b2	20

aa	av	bb	bv
a1	10	b1	b1
a2	20	b2	-

merge(lef, right, how='inner', on='key')

5. Reshapping Data:

The process of making that all the array value to be get Reshapped.

```
import numpy as pn
a = pn.arange(8).reshape(2, 4);
```

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

6. Aggrigation:

- used to apply some aggregation across one or more column.

```
import pandas as pd
a = {'a': pd.Series([10, 20, 30]),
     'b': pd.Series([1, 2, 3])}
b = pd.DataFrame(a)
b.aggregate(['sum', 'min'])
```

O/p:

Sum:	60	6
min:	10	1

7. Grouping Data:

- is used to split the data into groups.

```
import pandas as pd
a = {'a': pd.Series(['a', 'b', 'c']),
     'b': pd.Seri4([1, 2, 3]),
     'c': pd.Series([2, 3, 4])}
d = pd.DataFrao(a)
d.groupby(['a'])
```

Interacting with DataBases:

1. Interact with DataBase in text formate
2. Reading data from text formate.
3. writing data to text formate.

Interacting with data in Binary formates:

1. HDFS.
2. Interacting with Data in mongo DB.

Interacting with data in Redis:

1. The simple value.
2. List
3. set
4. oriented set.

UNIT-5:

Data visualition:

5.1 The matplotlib API primary line properties
5.2 Figures and subplots.
5.3 Explain planer plot types.
5.4 Scatter plots.
5.5 Bar plots.
5.6 Histogram plots.
5.7 Legends and annotations.
5.8 plotting functions with pandas
5.9 Addition python data visualition tools.

Bokeh

Mayalci

a	b	c
a	1	2
b	2	3
c	3	4

1. Interacting with DataBase.

Interacting with DataBase that is SQL DB.

Download & install:

install mysql-connector-python.

Test MySQL Connector:

```
import mysql.connector
```

Creating a Connection:

```
import mysql.connector
db = mysql.connector.connect(
        host = "localhost",
        user = "yourusername",
        password = "yourpassword")
print(db)
```

Create a DataBase:

```
import mysql.connector
mydb = mysql.connector.connect(
        host = "localhost",
        user = "username",
        password = "passdown")
mycursor = mydb.cursor()
mycursor.execute("~~Create DB~~")
        "CREATE DATABASE mydatabase")
```

Output:

c:\Users\My Name> python demo_mysql-create_db.py

dbname

```
for x in mycursor:
    print(x)
```

Creating a table:

we can create table with "CREATE TABLE" stmt

```
import sql.connector

db = mysql.connector.connect(
        host = "localhost",
        user = "user",
        password = "password",
        database = "my database"
    )

mycur = db.cursor()
mycur.execute("CREATE TABLE Students
        (name varchar(25),
        pin int)")
```

Inserting Data into Table:

```
sql = "INSERT INTO students (name, pin)
        values (%s, %d)"
val = ("Ram", 10)
mycur.execute(sql, val)
mydb.commit()
print(mycur.rowcount, "record inserted")
```

O/p:

1 row inserted

Select Statment :

```
mycur.execute("SELECT * FROM Students")
res = mycur.fetchall()
for x in res:
    print(x)
```

o/p: Ram 10

Where Statment:

```
Sql = "SELECT * FROM Students WHERE Sid='102'"
mycur.execute(Sql)
myres = mycur.fetchall()
for x in myres:
    print(x)
```

OrderBy : Sort ascending by default

```
Sql = "SELECT * FROM Students ORDER BY Sid"
mycur.execute(Sql)
myres = mycur.fetchall()
for x in myres:
    print(x)
```

DESC

Deleting a Record:

```
Sql = "DELETE FROM Customers WHERE Sid='102'"
mycur.execute(Sql)
myres = mycur.fetchall()
for x in myres:
    print(x)
```

Drop a Table:

```
sql = "DROP TABLE Students"
mycursor.execute(sql)
```

update:

```
sql = "UPDATE students SET sid = '102' where
                                sid = '101' "
mycur.execute(sql)
myres = mycur.fetchall()
for x in myres:
        print(x).
```

Limit Getting:

```
mycur = mydb.cursor()
sql = "SELECT * FROM Studen 3 LIMIT 5 "
mycur.execute(sql)
myres = mycur.fetchall()
for x in myres:
        print(x)
```

Join two tables:

```
sql = "SELECT users.name AS user,
              products.name AS favorite
              FROM users
              INNER JOIN products ON
              user.fav = product.id"
mycur.execute(sql)
myres = mycur.fetchall()
for x in myres:
        print(x)
```

8. Reading text lines in a python:

open(filename, mode)

read (r)
write (w)
append(a)

close( )

we can read by the help of:

read()
readline()
readlines()

1. read()

```
// f=open('aapy.txt ",r) //
with open('aapy.txt') as f:
        con = f.read()
        print(con)
```

2. readline()

```
with open('aapy.txt') as f:
        li = f.readline()
        while li:
                li = f.readline()
                print(li)
```

3. writes text line in a python:

To write the code like this.

```
data = [ ['aa', 'bb', 'cc'],
         [ 1,   2,   3],
         [ a1, b2, c3]]

with open("abc.txt", "w") as f:
    for i in data[1:]:
        aa, bb, cc = i
        line = "{} is a a {} is a b {} c\n".
                 format(aa, bb, cc)
        f.write(line)
```

3. Interacting with DataBase in Binary Format HDFS.

Hadoop File System

Connecting to HDFS Data:

Host : set this value to the host of your HDFS installation

Port : set this value to the port of your HDFS installation, Default port : 50070.

Code:

```
import petl as etl
import pandas as pd
import cdata.hdfs as mod

con = mod.Connect("Host = sandbox-hdp.hortonworks.com;
                  Port = 50070; path = /user/root;
                  user = root;")

Sql = "SELECT Field, Children From File Where
       Filed = '119116'
```

Interacting with mongodb.

mongo db stores data in JSON

- Here we use the PyMongo

instal pymogo

Create DataBase called mydatabase.

```
import pymongo as pya
cli = pya.MongoClient("mongodb://local host:27017/")
db = cli["mydatabase"]
```

Creating a Collections:

```
mycli = pyo.MongoClient('mongodb://localhost:27017/')
db = cli['mydatabase']
col = db["Students"]
```

Insert:

```
import pymongo as pym
cli = pym.MongoClient("mongo://localhost:27017/")
db = cli["mydatabase"]
col = mydb["Customers"]
dict = {"name": "John", "age": 29}
x = col.insert_one(dict)   // Single
li=[
    {"aa":10},
    {"bb":20},
    {"cc":30}]

x = col.insert_many(li)
```

Find:

```
myx=col.find_one()
print(x)                    // find one

for x in col.find():
    print(x)
```

Query:

used to filter Result by find()

```
myquery = {"sname": "aa"}
mydoc = mycol.find(myquery)
for x in mydoc:
    print(x)

myquery = {"sname": {$regex": "^s"}}
```

↳ str for S adder

Sort:

```
mydoc = col.find().sort("sname")
for x in mydoc:
    print(x).                   // ascel

mydoc = col.find().sort("sname", -1)
                                // descel
```

Delete:

```
mycol.delete_one(myq)
mycol.delete_many(myq)
```

Drop Collection:

```
col.drop():
```

mongoDB update:

```
mq = {"add", "Adv"}
nv = {"$set": {"add": "NITM"}}
mycol.update_one(mq, nv)
mycol.update_many(mq, nv)
```

MongoDB limit()

```
mr = mc.find().limit(5)
for x in mr:
    print(x)
```

6. Interact with DB Redis:

It is pronounced as Redis. = RED-iss
REE-diss
Red-DEES

Install Redis,

```
pip install redis.
```

or

```
import redis.
```

Run Redis Server.

```
$redis server.
```

Sample code with values.

```
import redis
portr = 6379.
hostr = 127.0.0.1
r = redis.Redis(host = REDIS_HOST hostr,
                port = portr)
print(r.ping())
print(r.key())
r.set('key1', 'value1')
print(r.get('key1'))
```

Redis with list:

```
rc = redis.StrictRedis(host='localhost',
                       port=6379,
                       db=0)

rc.lpush('LL', 'Java')
rc.lpush('LL', 'C')
while(rc.llen('LL') != 0):
    print(rc.lpop('LL'))
```

Redis with set:

a. add
b. remove elements

a. add elements:

```
rc = redis.StrictRedis(host='localhost',
                       port=6379,
                       db=0)

colorset = "colour"
rc.sadd(colorset, "Red")
rc.sadd(colorset, "Blue")
rc.sadd(colorset, "Green")
print(rc.smembers(colorset))
```

b. remove elet

```
rc.srem(colorset, "Blue")
```

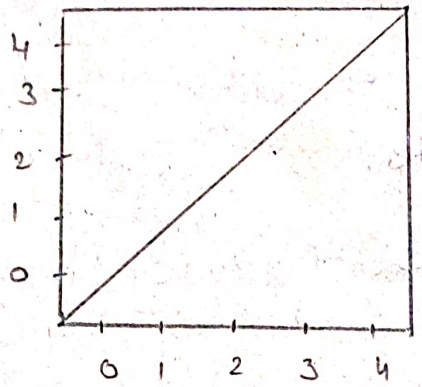
5. Data Visualation

Matplotlib:

1. It is a low level graph plotting library in a python that serves a visualition data.
2. It is done by John D Hunter.
3. It is a open source we can easily access.

Get started:

```
import matplotlib
```



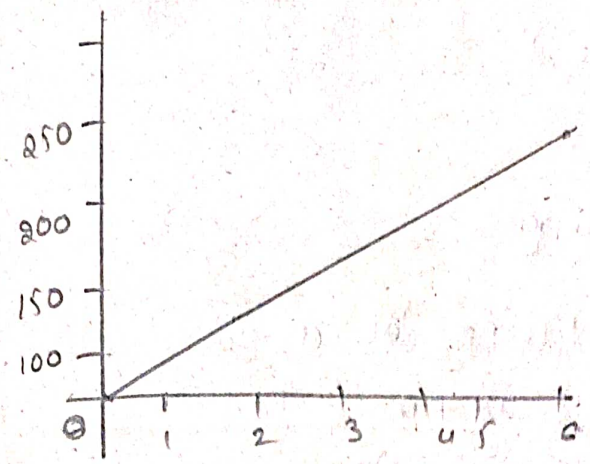
Matplotlib Pyplot:

Most of the Matplotlib utilities lies under the "pyplot" sub.module, and are usually imported under "plt" alias.

Example:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0,6])
y = np.array([0,250])
plt.plot(x, y)
plt.show()
```

output :



~~Matplotlib Mapping~~

Matplotlib plotting;

plotting of x and y points.

1. The plot() function is used to draw points (marks) in a Diagram.

2. By default, the plot() function draw lines from point to point.

3. The function takes parameters for specifying points in a diagram.

4. Two parameter we pass

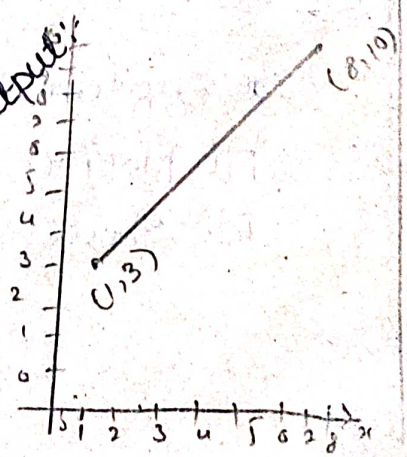
a. parameter-1 → x-axis.

b. parameter-2 → y-axis.

Code:

```
import matplotlib.pyplot as pt
import numpy as np
x = np.array([1, 8])
y = np.array([3, 10])
pt.plot(x, y)
~~point~~ pt.show()
```

output:



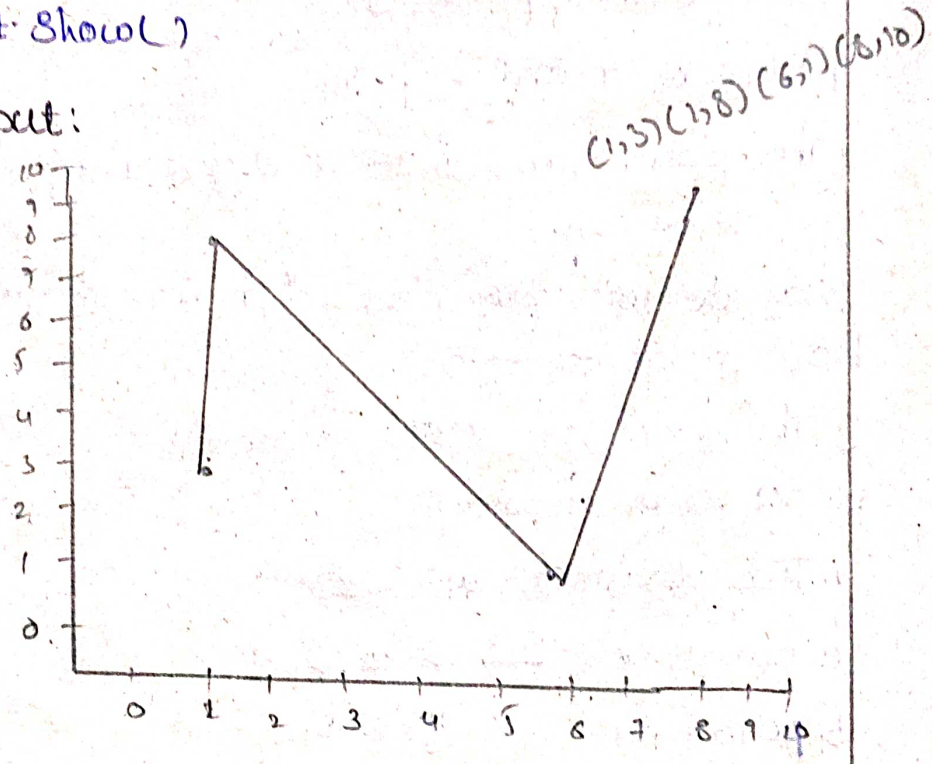
plot without Line:

```
plt.plot(x, y, 'o')
plt.show()
```

Multipoint plotting:

```
import matplotlib.pyplot as pt
import numpy as np
x = np.array([1, 2, 6, 8])
y = np.array([3, 8, 1, 10])
plt.plot(x, y)
pt.show()
```

Output:

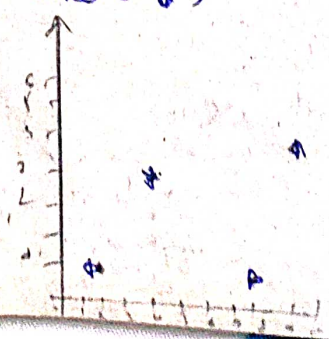


Multipoint Markers:

we use the keyword marker argument

marker

```
y = np.array([2, 4, 8, 10])
plt.plot(x, y, marker = '*')
plt.show()
```



Matplotlib Line:

use keyword argument linestyle or shorter ls, to change the style of the plotted line.

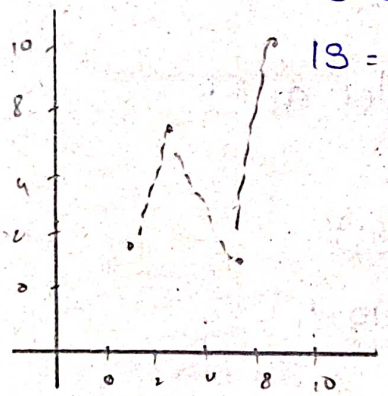
Ex:

```
import matplotlib.pyplot as plt
import numpy as np
y = np.array([3,8,1,10])
plt.plot(y, linestyle='dotted')
plt.show()
```

dotted .....

dashed ----

ls = ':'



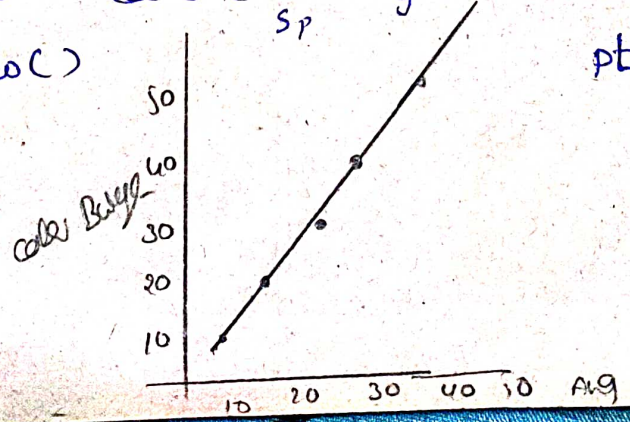
Matplotlib Labels and Titles.

we have the use the xlabel() and ylabel()

```
import matpotlib.pyplot as pt
impost numpy as n
a = n.array([10,20,30,40,50])
b = n.array([10,20,30,40,50])
pt.plot(a,b)
pt.xlabel("Average plus")
pt.ylabel("Calorie Burnage")
pt.show()
```

pt.title("SP")

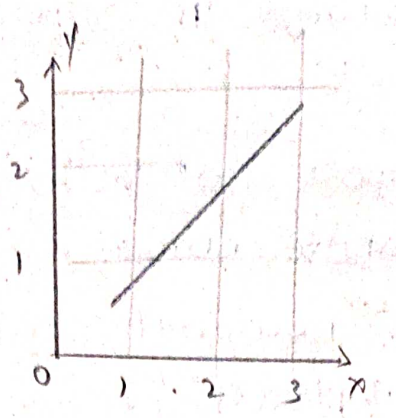
Output:



Matplotlib adding Grid Lines:

By using grid() function to add grid line to the plot.

```
import numpy as np
plt.grid()
```



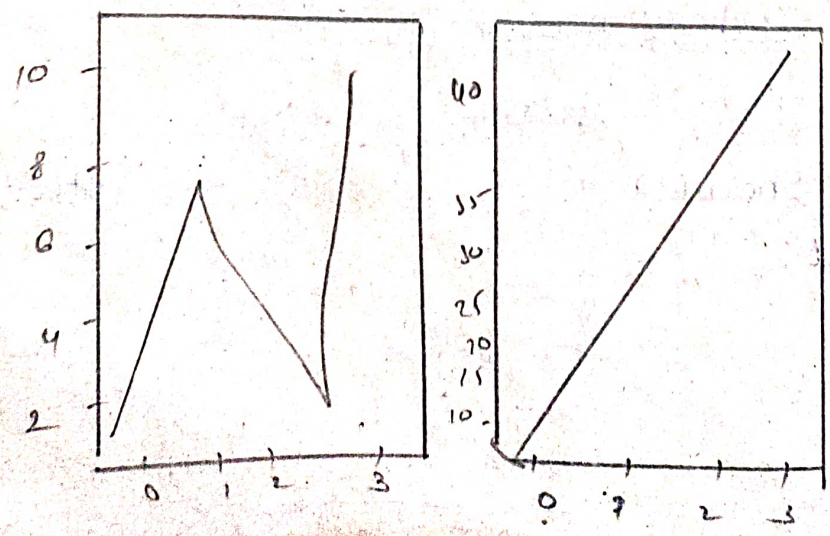
Matplotlib Subplots:

using subplots()

Code:

```
import matplotlib.pylot as pt
import numpy as np.

x = np.array([0,1,2,3])
y = np.array([3,8,1,10])
pt.subplot(1,2,1)
pt.plot(x,y)
x = np.array([0,1,2,3])
y = np.array([10,20,30,40])
pt.subplot(1,2,2)
pt.plot(x,y)
pt.show()
```

Output:

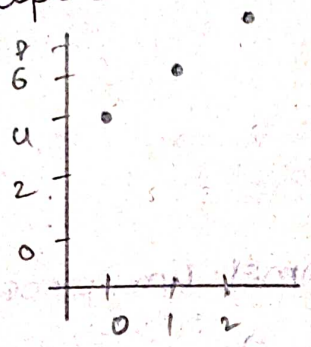


matplotlib scatter:

By using scatter()

```
x = np.array([0,1,2])
y = np.array([4, 6, 8])
pt.scatter(x, y)
pt.show()
```

output:

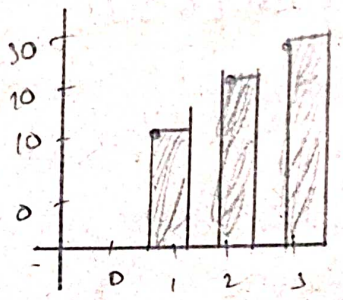


matplotlib Bars:

bar()

```
x = np.array([1, 2, 3])
y = np.array([10, 20, 30])
```

pt.bar(x, y)

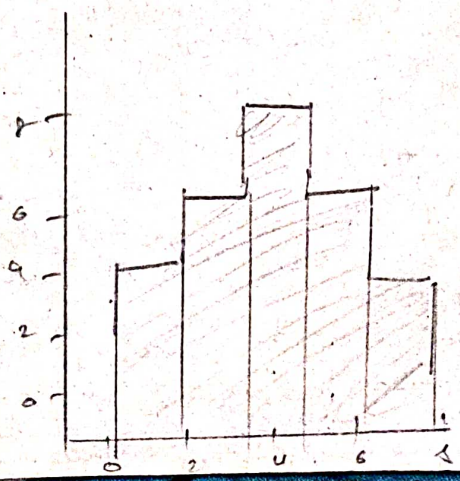


barh( ) - horizontal

matplotlib histograms:

It is a sequence of Distribution.

```
x = np.random.normal(170, 10, 250)
pt.hist(x)
pt.show()
```

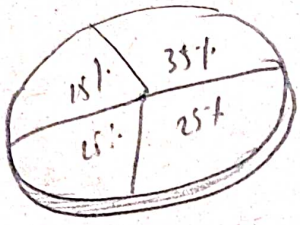


matplotlib. pie charts:

That can be done by pie( )

```
y = np.array([35, 25, 25, 15])
pt.pie(y)
pt.show( )
```

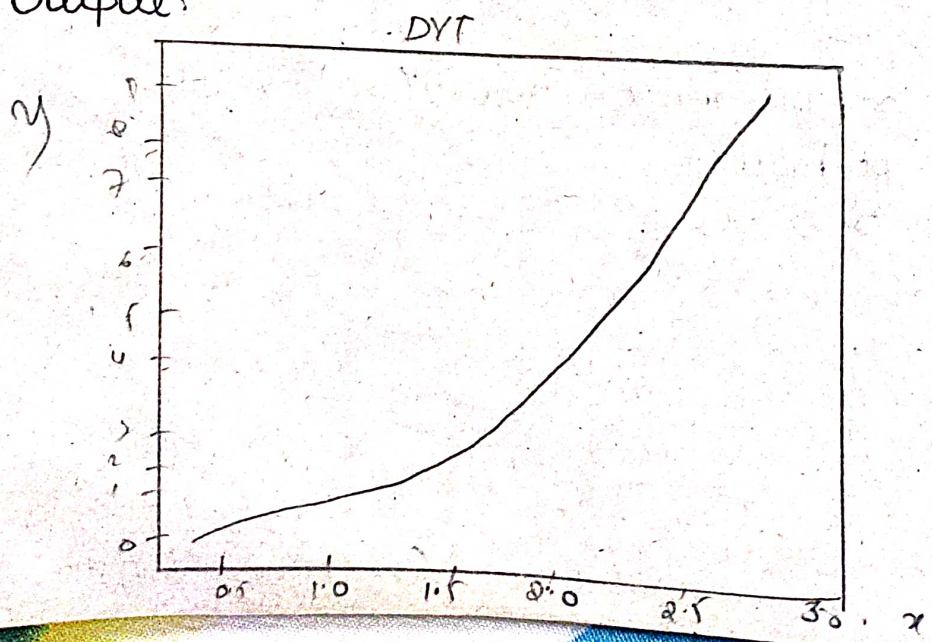
Output:



5.1 The matplotlib API primes line properties.

```
import matplotlib.pyplot as pt
import numpy as np
x = linespace(0,3,6)
print(x)
y = np.array([0., 0.6, 1.2, 1.8, 2.4, 3.0])
figure( )
plot(x,y)
xlabel('x')
ylabel('y')
title(" DVT")
plt.show( )
```

Output:



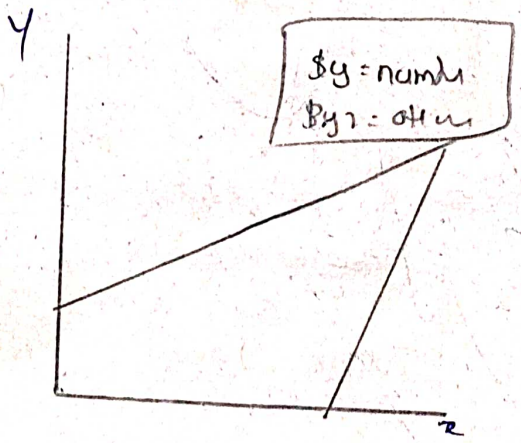
52 plotel types:

- Scatte
- histogram
- Bar
- pie
- and etc...

53 Legands: and Annotations:

Legands:

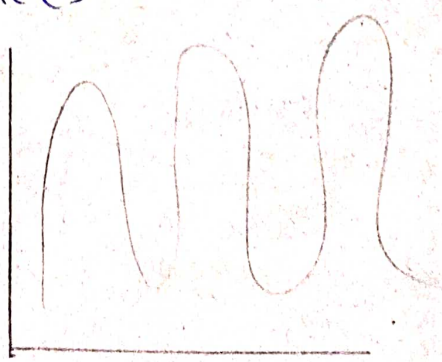
```
x=[1,2,3]
y=[1,2,3]
plt.figure()
ax.plot(x,y, label='$y = Numbe')
ax.plot(x,y1, label='$y = other')
plt.title('leged inside')
ax.legend()
pt.show()
```



Annotate plot:

text() annotate()

```
ax.annotate()
```



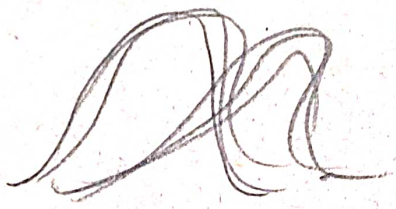
5.8 plotting functions with pandas.

```
import matplotlib.pyplot as pt
df.pt(x="Rank", y=["path", "median", "path"])
pt.show()
```

Code:

```
import matplotlib.pyplot as pt
import pdandas as pd
df = pd.read_csv('path')
    pd.plotting.andrews_curves(df, 'Name')
```

o/p:



The End