20/9/21

→ Python is introduced in 1991 by G. Van Rossom

→ Python is General programming Language

→ Simple

→ Interpreted programming Language (No compilation and ready for run)

→ High Level Language (Basic English written)

→ Object oriented programming Language (class, object)

```
struct student
{
    char n[30];
    int a;
};
struct student s1;
```

class (collection of objects)

C

code

↓ Compilation

Machine code

↓

Ready for Run

Python

code

↓

Ready for Run

Object orientated programming (oop):-

→ Python supports object orientated programming.

→ oop represented in terms of classes and objects.

→ The basic features supported by oop is

1) Class (logical)
2) Object (class variable)
3) Method (sub part of class) (Perform a specific task)
3) Data abstraction
4) Data encapsulation
5) Inheritances
6) polymorphism

21/9/21

Class :-

→ It is a user defined data type.

→ It is a collection of data members (variables) and methods (It is used to perform a task)

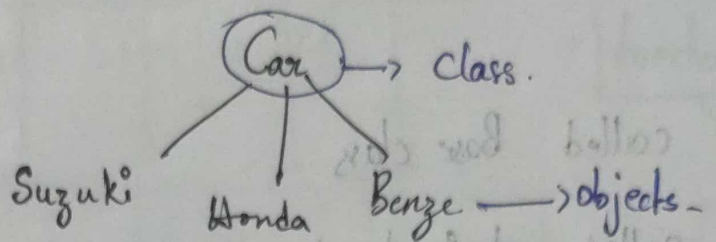
→ A class is a logical structure which is not physically exist.

→ To allocated the memory to the class we have to create a class variable which is called as object.

Object:-

→ A class variable is called object. (or)

→ instants of a class is called object.



Data abstraction:- The process of using the functionality's without knowing the background detials is known as Data abstraction.

Ex:- 1) usage of mobile. How internal it works.

2) usage of predefined functions in programming languages without knowing How internal it works

Data encapsulation:- The process of combinning the data and methods under in a single unit is called Data encappulation. Using this we can achive data. hiding.

data hiding can be achive by using protected (or) private acesers.

***

Inheritaries:- The process of creating new class from the existing class is called inheritance.

Existor A.

↓

Newclass B

Existing class is called Base class

New class is called derived class.

Polymorphism :— Poly = "many" morphism = "forms"

One thing performing multiple forms is called polymorphism

→ Python is supporting concept of polymorphism through method over load.

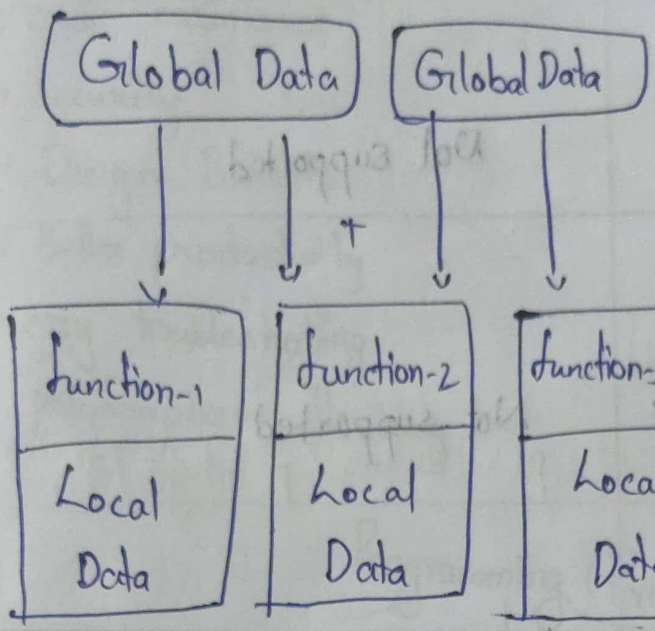
Ex:- human being behaviour at different situations like with friends, with family, with in the class.

Difference blw oop & POP

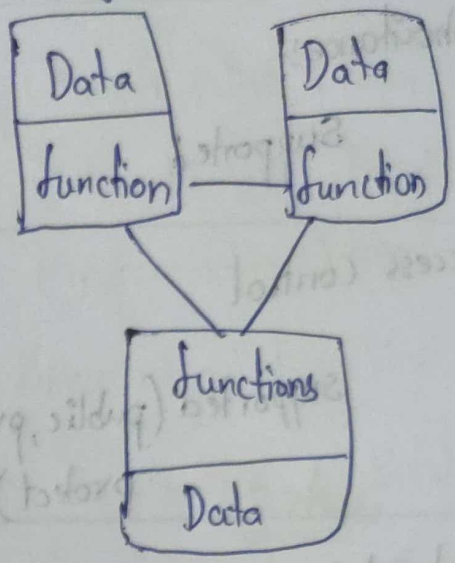
The key difference blw object & Procedure

→ The main diff blw oop & pop is oop divides the program into small objects & pop divides the program into functions.

Procedural oreinted Programming



Object oriented Programming



Parameters	OOP	POP
Basic Definition:-	Oop is object oriented	Pop is procedural or sturcture oriented
Program Division:-	The program is divided into objects	The program divided into functions
Approach:-	Bottom up approach	top-down approch
Data Control:-	Data in each objed is controlled on it's own	Every function has different data, so there's no controlled over it.
Msg passing:-	Object functions are linked through msg passing	Parts of a program are linked through parameter passing.
Expansion/Adding data:-		

Adding new data is easy	Adding new data is difficult.
Inheritance:- Supported	Not supported
Access control supported (public, private protect)	Not supported
Data hiding:- Data hiding is achive by using encapsulation.	No data hiding
polymorphism Supported	Not supported
Code reuseability Supported	Not supported
type of problems:- using for big problems	using for small problems
Ex:- c++, java, python	Ex:- C, Forten, Pascal.

Advantages of oop:-

→ code reusability
→ code maintance
→ Security
→ Design Benifits
→ Better productivity
→ easy troubleshotting
→ polymorphism Flexibility

27/9/21

Programming (Problem).

Input — Logic — Output

taking the data

1,
a=3
b=4

a,b=3,4

Reading the data.

↓

input()

↓

input("msg")

↓

String

integer — Float

int(input("msg")) float(input("msg")).

Write a python code to Perform the formulae.

1. Reading 3 integers

x = int(input("enter no 1"))
y = int(input("enter no 2"))
z = int(input("enter no 3"))

2. Write a python code to read biodata (name, branch, Section then mobile no)

```
a = input("enter name")
b = input("enter branch")
c = input("enter section")
d = int(input("enter mobile.no"))
```

3. Write a python code to read ptr values. (p=integer, t, r=float)

```
p = int(input("enter no"))
t = float(input("enter no"))
r = float(input("enter no"))
```

Output Statement :- This statement is used to display the data on screen (python prompt).

-> To implement this we are using a method print.

msg :- print("msg")

value :- print("variable name")
print(a)

msg + value :- print("msg", v1)
print("msg", v1, "msg", v2)

Display 3 integers.

```
print("x= ", x, ", y= ", y, ", z= ", z)
```

(2)

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

(3)

```
print("p= ", p, ", t= ", t, ", r= ", r)
```

Introduction to python Environment:-

→ In python environment the most imp role by spaces (Intendention)

→ If code is not maintding proper intendation it is an error

Ex:-
```
print("hello")
print("bye")
```

```
print("hello")
print("bye")
```
} Error

Write a code program to find the addition of three number.

Logic :-

Input :- x, y, z

Logic :- ad = x+y+z

Output :- ad

Code:-

```
x = int(input("enter no"))
y = int(input("enter no"))
z = int(input("enter no"))
ad = (x+y+z)
print("ad = ", ad)
```

Python Basics:— (token)

Keyword → Predefined words → interpretor (Translator)
if, else, elif, for...
Compiler, interpretor.

↳ Predefined word for a translator.

Variable → a=33 → 33 (a)

name = value

variable → to store the value/data.

Ex:- a=44 → int
a=1.44 → float
a="abc" → string

Assigning different values to different variables :—

a=33
b=44
c=55

a,b,c = 33,44,55

Common value to multiple variable:-

Name → (Alphabets, digits, _)

name1 = name2 = name3 = value

1) Cannot start with digit
2) white spaces, symbol.
3) Cannot be a keyword → if = 33.

↳ The type of variable decided during the execution the type of variable assign

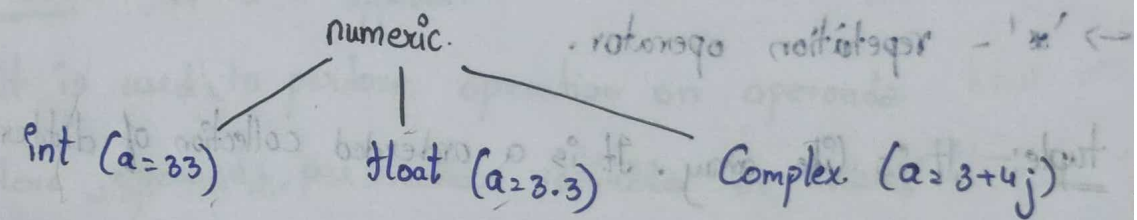
Types of data types :-

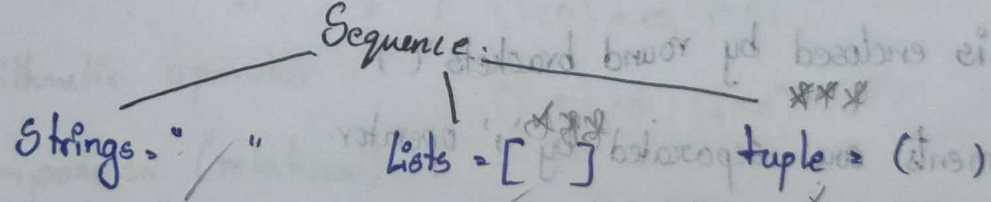
→ It is used to specify the type of data.

→ Python is a dynamically typed language. (here no need to define type of variable, the interpreter internally fixes the types depend up on the value assign)

→ Types :- Python is supporting the following types.

1) Numeric type
2) Sequence type.
3) dictionary type
4) Boolean type → True (or) False
5) Set type.





Sequence :- It is a collection of different types of data elements.

There are 3 types
1) Strings
2) Lists
3) tuple

String :- It is a sequence of characters it can be represented

in either single, or double or triple. code.

Ex:- a = "pravalika".
print(a)
pravalika.
type(a)
<class 'str'>

→ display one string in multiple times

```
s1 = "abcd"
s2 = "pqrs"
print(s1 * 2)
abcdabcd
```

```
s1 = "abcd "
print(s1 * 2)
abcd abcd
```

→ There are two operators '+' & '*' concatination (it is used to combine the strings.

→ '*' - repetation operator.

tuple:- It is like array. It is a ordered collection of different data.

→ It is enclosed by round brackets '( )'

→ elements are separated by ',' operator.

Syntax:- variable name = $(v_1, v_2, \ldots)$

→ element index count starts from 0.

Ex:- a = (1, 2.3, "rrr")
print(a[1]) → 2.3
print(a[2]) → rrr

Lists:- It is like array in c. It is a ordered collection of data.

→ It is enclosed by Square brackets [ ].

→ Elements are separated by ',' operator.

Syntax:- Variable name = [v_1, v_2, v_3]

c) element index count starts from 0.

Ex:- a = [1, 2.3, "rrr", (4+8j)]

```
print(a[1])
2.3
print(a[3])
(4+8j)
print(a)
[1, 2.3, 'rrr', (4+8j)]
```

Operators:- It is a symbol.

→ It is used to perform operation on operands

→ Here operands are either variables or values

→ There are 7 types of operators.

1. Arithmetic operator
2. Comparison (relational)
3. Logical
4. Assignment
5. Bitwise
6. membership
7. Identity.

Comparison operators:-
(Relational)

↳ It is a binary operator

↳ It is used to find the relational between operands.

↳ It always produces the result as either True (or) False.

a=5	b=7	
>	a>b	False
>=	a>=b	False
<	a<b	True
<=	a<=b	True
==	a==b	False.
!=	a!=b	True

logical operators:-

↳ To combine two (or) more relational expressions logical operators are used.

↳ logical operators also produce result as either True (or) False.

and	(a>b) and (a>c)
or	(a>b) or (a>c)
not	not(a>b)

Assignment operator:-

↳ It is used to assign the value to a variable.

variable name = value (or) expression

a = 10

S = 1 to 5

Shorthand assignment:-

a = a+30 => a+=30

a = a/10 => a/= 10

a = a*20 => a*=20.

Bitwise operator:-

→ This operators are used to pedorm operation on bits.

		I_1	I_2	&	|	^
&	bitwise and	0	0	0	0	0
|	bitwise OR	0	1	0	1	1
^	bitwise XOR	1	0	0	1	1
~	one's complement	1	1	1	1	0
<<	~~Right~~ Left shift (nox 2^n) (n = no. of bits)					
>>	~~Left~~ Right shift					

a=25 b=13

a&b
```
  00011001
  00001101
  --------
  00001001  =9
```

a|b
```
  00011001
  00001101
  --------
  00011101  => 29.
```

a^b
```
  00011001
  00001101
  --------
  00010100  => 20
```

1's complement (or) bitwise not

→ This operator is used to convert one's into zero's and zero's into one's in the given data.

Ex:- a=13 = 00001101

~(a) => 11110010

Bit Shift :-

→ It means rem. deleting the bits in the left end and add that no. of bits(0) in the right end.

→ $no \times 2^n$ [n = no. of bits]

→ a=13
print (a<<2)
52
print (a<<4)
208.

Right shift:- It means deleting the bits in the right end and add that no. of bits (0) to the left end

→ a=13
print (a>>2)
3

13 => 00001101
00000011
=> 3

=> $no // 2^n$ (n = no. of bits)

00011000
10000
10111000

00011000
10110000
10010000

Membership operator:-

This operator is used to check the particular element is available in the specified object (specified list of elements)

a = [1, 2, 3, 4, 5]
print (3 in a)
True.
print (9 in a)
False

print (3 not in a)
False
print (9 not in a)
True

→ It is used to perform searching activity.
→ There are 2 operators

in → existance (yes → True, no → False)

not in → not existance (yes → True, no → False.)

Identity:-

→ Identity operator is used to identify the type of variable (or) element (or) an object.

There are 2 operators is → type. (Yes → True, no → False)

is not → type (Yes → True, no → False)

```
a=3
print (type(a) is int)
True
print (type(a) is not int)
False
a=[1,2,3,4,5]
print (type (a) is list)
True
print (type (a) is float)
False.
print (type(a) is not float)
True.
```

Expression:-

expression is in the combination of operators and operands.

It is always produce the result as value.

If an expression contains more than one operation then evaluation takes place as per the operators precedance.

There are 5 types of expressions operators

1) arithmetic. Ex:- a+b

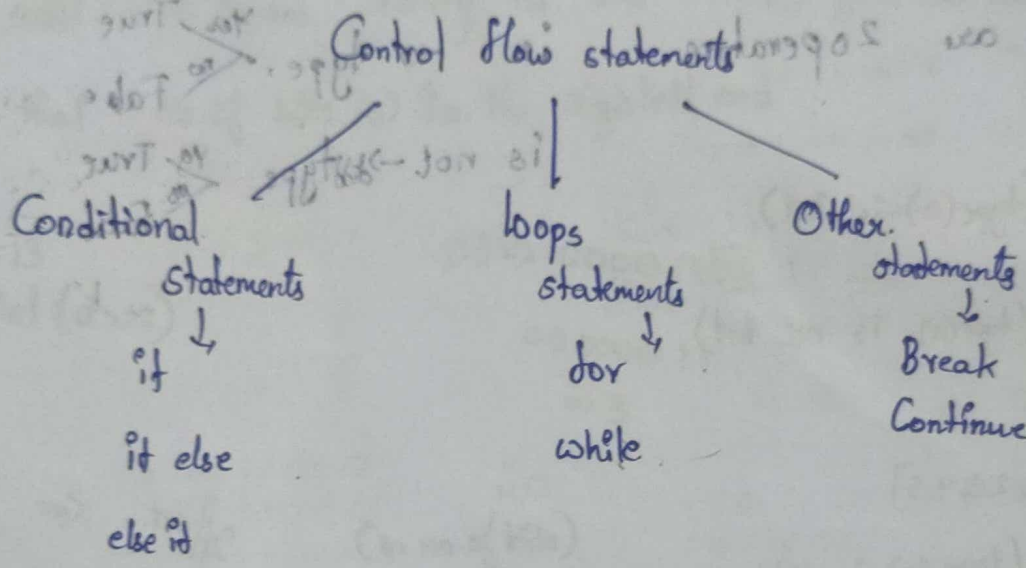
2) Relational. Ex:- a>b

3) logical. Ex:- (a>b) and (a>c)

4) Bitwise. Ex:- (a<<2) (or) (a>>4)

5) constant Ex:- 3+6

Control flow statements:-



→ These statements are used to control the flow of execution of a program.

Conditional Statements:- These statements are used to perform a specific task based on condition.

1) If statement:-

Syntax:- if (condition):
    st1
    st2
    st3
    stn

Ex:- a=33
if(a==33):
    print("equal")

→ This statement is used to perform a task, when the condition is true.

if else statement:-

This statement is used to perform a task on either the true (or) false cases.

Syntax:-
```
if (condition):
    st1
    st2
else:
    st1
    st2
```

Ex:-
```
a=int(input("enter no"))
b=int(input("enter no"))
if(a>b):
    print("a is big")
else:
    print("b is big")
```

Output:- enter no 24
enter no 55
b is big.

else if ladder:- It is used to work with the multiple conditions.

⇒ When a program involved to perform different task on different conditions

Ex:-
```
a,b,c,d=10,22,150,33
if(a>b) and (a>c) and (a>d):
    print("a is big")
elif (b>c and b>d):
    print("b is big")
elif (c>d):
    print("c is big")
else:
    print("d is big")
```

NOTE:- else block is not mandatory

Loop Statements:- These statements are used to perform a task repeatedly.

for - auto updating. for variable in range( ):

while:- range(10) - 0 to(10-1)
range(1,10)
range(1,10,2).

Unit-2 (Module-2).

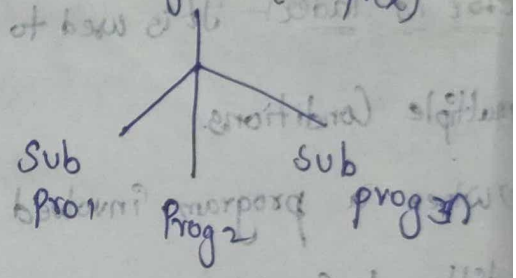
Functions:-

→ It is collection of one or more statements to perform a task.

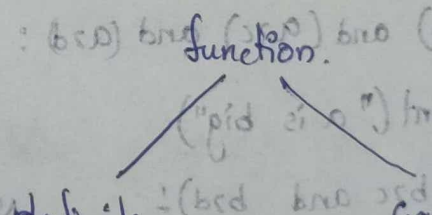
→ A programming can be divided into sub-program that sub program is also called as function

Advantages:-

1) Easy to understand
2) Code Reusability
3) Error Rectification. is easy.

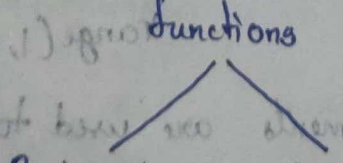


→ To create a function in python (def) keyword is used.



def functionname (parameters):
optional.

Types of functions:-



def function name (parameters):
Body of functions
return(value/Exp)

Cor.

functionname( )

Working functions:-

1) defining the function
2) calling the function.

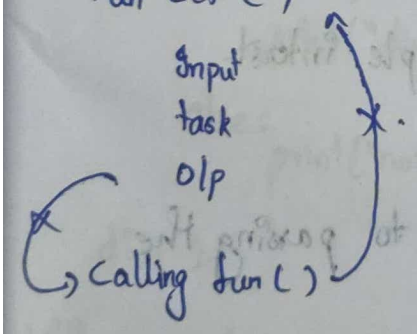
Ex:- Average of 2 numbers.

```
def avg ( ):
    x = int(input("enter no = "))
    y = int(input("enter no = "))
    s = (x+y)/2
    print("Avg = ", s)
avg( )
```

Categories of functions:- There are 4 catagoeirs.

P → Parameters R → Return.

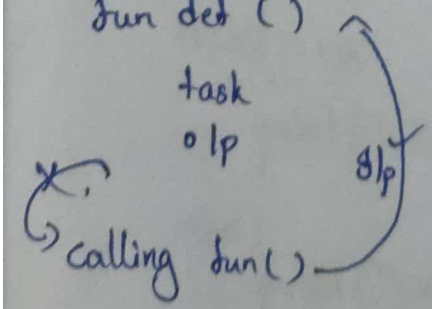
1. WOP WOR.



Ex:-

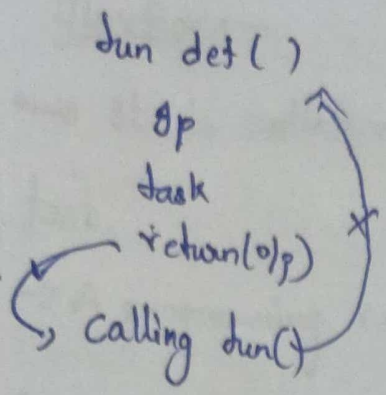
```
def sum ( ):
    x = int(input("enter = "))
    y = int(input("enter = "))
    i = x+y
    print("sum = ", i)
sum( )
```

2. WP WOR



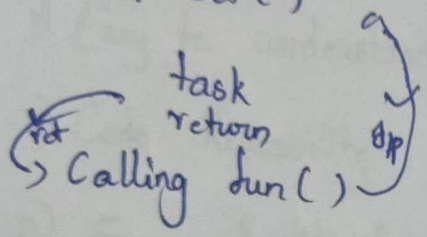

```
def sum (x,y):
    i = x+y
    print("sum = ", i)
sum (4,5)
```

3. WORP WR




```
def sum ( ):
    x=int(input("enter="))
    y=int(input("enter="))
    i = x+y
    return (i)
j= sum( )
print("sum=",j)
```

3. WP WR




```
def sum (x, y):
    i= x+y
    return (i)
j=sum(4,5)
print("sum=",j)
```

1) Write a program to check a given number is divisible by 7 or not.

2) Write a program to find the simple interest.

Passing arguments :- There are doure ways to passing the arguments

1) Required arguments
2) Arbitary
3) Keyword
4) Arbitary keyword.
5) Default

Required Arguments:- fun defintion & fun call must have the same no. of arguments.

Aribitary arguments:- when we don't know how many parameter passed to a function we can use arbitary argument.

Syntax:- * Parameter name.

→ It accepts all parameters as a ~~tuple~~ (tuple).

```
def cal (*p):
    print (p)
cal (2,5,78, "def")
```

o/p:- (2,5,78,'def')

1) without Parameter & without Return.

```
def div ( ):
    a=int(input("enter no = "))
    if(a%7==0):
        print("divisible")
    else:
        print("not divisible")
div( )
```

o/p:- enter no = 21
divisible.

With parameter & without Return

```
def div(a):
    if(a%7==0):
        print("divisible")
    else:
        print("not divisible")
div(21)
```

o/p:- divisible.

Without Parameter With Return.

```
def div( ):
    a=int(input("enter no= "))
    if (a%2==0):
        print ("divisible")
    else:
        print("not divisible")
    return (a)
j=div( )
```

o/p:- enter no=21
divisible.

With parameter & with Return.

```
def div(a):
    if (a%2==0):
        print("divisible")
    else:
        print("not divisible")
    return (a)
j=div(21)
```

o/p:- divisible

Simple intrest:-

Without Parameter & without Return.

```
def si( ):
    p=int(input(" enter no = "))
    t=int(input(" enter no= "))
    r=int(input(" enter no= "))
    s=(p*t*r)/100
    print("si= ",s)
si( )
```

o/p:- enter no=10
enter no=20
enter no=30
si=60.0

Without Parameter & with return.

```
def si( ):
    p=int(input("enter no="))
    t=int(input("enter no="))
    r=int(input("enter no="))
    s=(p*t*r)/100
    return(s)
j=si( )
print("si=",j)
```

o/p:- enter no=10
enter no=20
enter no=30
si=60.0

With Parameter & without Return.

```
def si(p,t,r):
    s=(p*t*r)/100
    print("si=",s)
si(10,20,30)
```

o/p:- si=60.0.

si(t=10, t=30, p=20)

with parameter & with return.

```
def si(p,t,r):
    s=(p*t*r)/100
    return(s)
j=si(10,20,30)
print("si=",j)
```

o/p:- si=60.0

Keyword arguments:- Through calling function we can assign the values to the parameters by specifying the name of the parameters.

→ Here we can change the order of the parameters.

Ex:- def var (a,b,c):
print(a,b,c)
var (c=33, a=88, b=49)

o/p:- 88 49 33

Keyword Arbitrary Arguments :- When we don't the know the no. of keyword arguments passed to a function.

→ We can use keyword arbitrary argument

Syntax:- def var (** vaariable name):
| Body of function.

→ Here argument acted as a dictionary

Ex:- def var (**k):
print(k)
print (k.keys())
print (k.values())
var (c=10, b=33, a=44)

o/p:-

Default Parameters :- Python allows functions to create a default parameters.

→ Default parameters assigning followed from Right to left.

Ex:- def var (a, b=55, c=90):
    print (a,b,c)

var (22,30,40)
var(22,30)
var(22)

o/p:- 22 30 40
22 30 90
22 55 90

Recursion:- The process of calling a function itself is called as recursion.

→ A recursive function without conditional statements causes a infinite loop.

Ex:- def var ( ):
    print ("hi")
    var( )
    var( )

hi
hi
hi
.
.
.

Recursion work on data structure (stack).

def dun ( ):
normal Ex. { - - - - -
- - - - -
dun( ) // Recursive call.
stack { - - - -
- - - -

Ex-1:- Write a Recursive program to display a msg hi 10 times

```
def display (x):
    if (x>=1):
        print ("hi")
        display (x-1)
display (10)
```

hi
hi
hi
hi
hi
hi
hi
hi
hi
hi

```
def display (x):
    if (x<=10):
        print (x)
        display(x+1)
display(1)
```

o/p

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

Factorial:-

```
def facto (n):
    if (n>=1):
        return n*facto(n-1)
    else:
        return 1
print(facto(5))
```

without return

```
f=1
def fact(n):
    global f
    if(n>=1):
        f=f*n
        fact(n-1)
    else:
        print("fact=",f)
fact(5)
```

Variable:- There are two types of variable.

Scope of the variable:-

→ Scope defines the acessbility of a particalar variable.

↪ Two types of vaaibles

1) Local variable.

2) Global variable

Local variable:- A variable decbared inside a function.

called as Local variable

↪ It scope is with in the function

Here x is a variable it's scope within the var function.

↪

```
def var():
    x=44
    print(x)
var()
```

Global Variable :- A variable which is declared outside all function is called as global variable.

↳ In this x is a global variable.

```
x=33
def var():
    y=44
    print(x,y)
var()
```

↳ When we are declaring local variable with same of global variable, local variable overriding on global variable.

Ex.
```
x=33
def var():
    x=44
    print(x)
var()
```

(To use a global ~~variable within the func~~)

→ To modify a global variable within the function by using global key word

Ex:-
```
x=33
def var():
    global x
    x=44
    print(x)
var()
print(x)
```

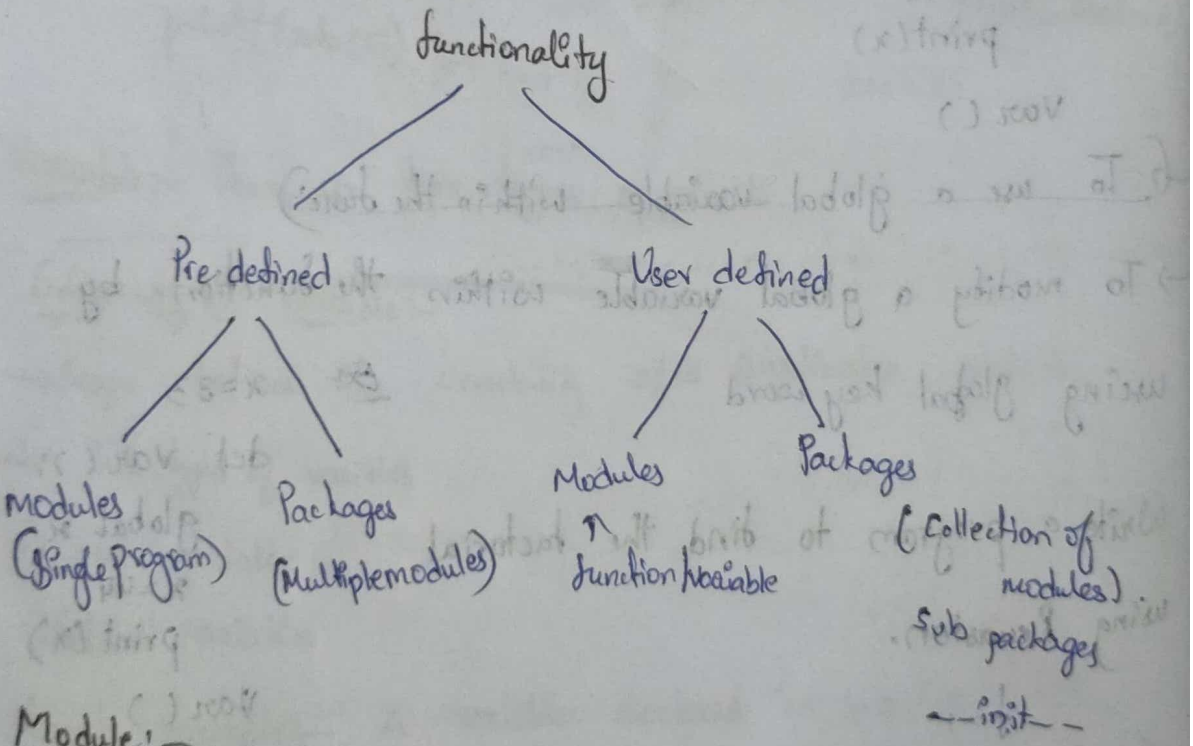
Write a program to find the factorial using Recursion.

```
f=1
def fact(n):
    global f
    if(n>=1):
        f=f*n
        fact(n-1)
    else:
        print("fact=",f)
fact(5)
```

Fibonacci Series:- Sum of perivous 2 terms = next term

```
n=55
f1=-1
f2=1
for i in range(n):
    f=f1+f2
    print(f, end=" ")
    f1=f2
    f2=f
```

```
n=8
def fib(f1,f2):
    global n
    if(n>=1):
        f=f1+f2
        print(f)
        f1=f2
        f2=f
        n=n-1
        fib(f1,f2)
fib(-1,1)
```



Module:-

1) A module is a file or a program containing set of functions

2) To create a module save the file with .py extension

3) To use the module in our program we can use the import statement,

Syntax:- import module name

→ To access the functionality of the module.

Syntax:- modulename.function name( ).

→ A module ~~taking~~ may contain variables (or) function.

→ ~~There are~~

→ We can rename the module using the following syntax.

Syntax import modulename as alternate name.

→ There are two types modules
1) Pre defined
2) User defined.

Ex:- Create of a module.

```
def f1():
    print("hi")
def f2():
    print("bye")
def f3():
    print("hello")
```

Save it with → abcd.py

Accessing a module:-

→ it is done by using import.

```
import abcd
abcd.f1()
abcd.f2()
abcd.f3()
```

o/p:- hi
bye
hello.

Number System:

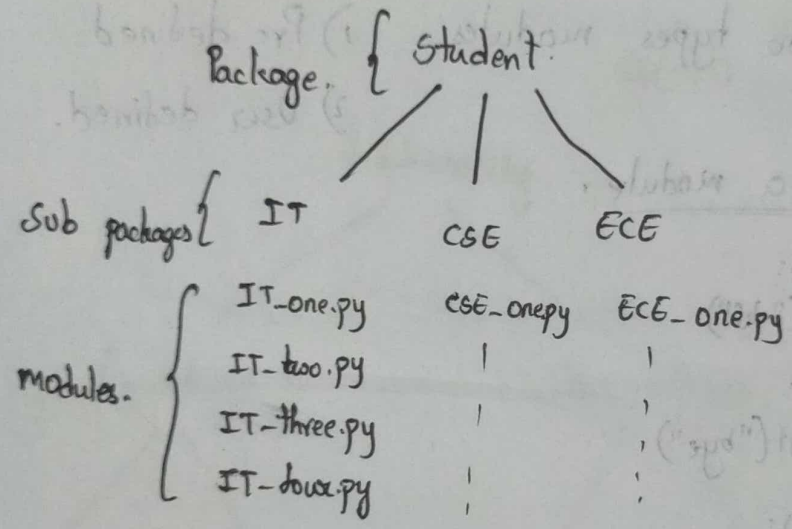
Packages :-

→ It is a directory having a collection of modules.

→ It contains a special file called __init__.py. It may contain sub packages.

→ We can access the packages by using import statement

→ Inside package functionality can be access by using module name . functionality name.



Ex:- Create a package student with 3 modules (IT, CSE, ECE) access it's functionality and display it.

Student

__init__.py

it.py → f1() f2()

cse.py → f3() f4()

ece.py → f5() f6()

1) it

```
def f1():
    print("hi")
def f2():
    print("bye")
```

2) def f3():
    print("hello")
def f4():
    print("bye")

3) def f5():
    print("hello")

def f6():
    print("people")

form student import it
it.f4()
it.f2()

o/p :- hi
bye

Difference b/w module & Packages.

Module	Packages
→ One program	→ collection of modules
→ no need of __init__.file	→ __init__.py
→ no need of directory files	→ need of directory files
→ no sub modules	→ sub packages.

Strings :-

→ It is a collection of character's.

→ Initialization of a string

→ strings are immutable.

→ A string can be initialized by using (" ") or (' ')

Ex :- s1 = "Apu"
print(s1)
s2 = "Sindu"
print(s2).

Multiline Declaration :-

```
S2 = ''' Pravalika.
Anitha Pranay.
Ramana '''.
Print(S2).
```

Reading a string from the keyboard :-

→ It is done by using input function.

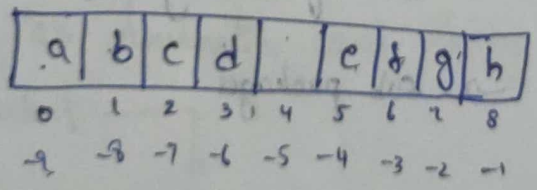
```
S1 = input (" enter a string = ")
print(S1).
```

→ For displaying string ~~funct~~ using print function.

→ Accessing a string :-

→ A string can be accessed by using index number followed by string name.

→ S1 = "abcd efgh"




```
S1 = "abcd efgh"
print (S1[3])
print (S1[-4])
```

o/p :- e
e

Accessing entire string :-

```
S1 = "pravalika"
for k in ~~range~~
    print (k, end=" ")
```

o/p :- pravalika.

```
s1="pravalika"
for k in range (9):
    print (s1[k], end=" ")
```

Slicing :-

↪ This is an operation to ~~abstract the~~ extract the substring.

It uses the following syntax.

stringname [enter index no : index number]

Ex:-

```
s1="pravalika"
print (s1[0:3])
print (s1[-9:-6])
print (s1[4:])
print (s1[-9:])
```

Basic functionality's on strings:-

1) length of a string :- To find the length of string "len( )" function is used

Syntax :- len(stringname)

Ex:- s1="pravalika"
print("length =", len(s1))

o/p => length = 9

2) String Reverse :-

Ex:- s1="pravalika"
print("reverse =", s1[::-1])

3) String copy :- It can be done by using assignment operator

Ex:- S_1 = "pravalika"
$S_2 = S_1$
print("copied =", S_2)

o/p :- pravalika

4) String Comparision :- It can be done by using double equal (==) operator.

Ex:- S_1 = "pravalika"
S_2 = "Sindhu"
print("Comparision =", ($S_1 == S_2$))

o/p :- Comparision = false.

String concatination :- It can be done by using '+' operator. to concatinate.

Ex:- S_1 = "karnati"
S_2 = "Pravalika"
$S_3 = S_1 + S_2$.
print("Concatination =", S_3)

o/p :- karnatiPravalika

Lower case :- This function is used to convert the string into lower case letters.

Syntax :- string name . lower ( ).

Ex:- S_1 = "K pravalika"
print("lowercase =", S_1.lower())

o/p :- k pravalika.

Upper case :- This function is used to convert the entire string into the upper case.

Syntax :- string name.upper()

Ex :- s1 = "K pravalika" o/p :- KPRAVALIKA.

print ("Uppercase =", s1.upper())

Replace() :- This function is used to replace an existing sub string with new substring.

Syntax :- string name.replace("sub string", "new string")

Ex :- s1 = "here"

print ("replace =", s1.replace("e", "i"))

print ("replace =", s1.replace("he", "the")

o/p :- replace = hiri

replace = there

↪ If sub string will not avaiable then it will return the orginal string without updation.

Count :- It gives the no. of occurance of sub string. in the given string.

Syntanx :- string name.count('sub string')

find() :- This function returns the index position of the sub string in the given string.

Syntax :- string name . find ('sub string')

→ It returns the index no of first occurance sub string

→ If sub string not there it will return -1.

Ex :- s1 = "henllo friend"

print ("find =", s1.find ('en'))

o/p :- find = 1

rfind() :- This function

→ It is like find function but it will return last occured sub string.

Syntax :- string name . rfind ('sub string')

slipt :- This function is used to slipts the string into sub string.

Syntax :- string name . slipt ('separator')

Ex :- s1 = "hello friend"

print ("slipt = ", s1.slipt (" "))

print ("slipt = ", s1.slipt ("e"))

o/p :- ['hello', 'friend']

['h', 'llo fri', 'nd']

// palindrom. coith strings

```
s = input("enter = ")
rev = s[::-1]
if (s == rev):
    print("palindrome")
else:
    print ("Not palindrome")
```

Sequences :— There are 4 sequences
1. tuple.
2. List
3. Set
4. dictionary.

tuple:- It is used to store multiple values in a single name.

→ It can contain different type of data.

→ It is a collection of elements which ordered & unchangable

→ tuple are return with round brackets '( )'.

Syntax:-

```
name = ('value', value. . . . . )
stin = ("Ahobitha", 18, "IT-A", "AU")
```

→ It allows the duplicate values.

→ tuple item are indexed the first item starts with zero.

→ tuple Length:- It finds how many items a tuple has.

→ It can be calculated using len (tuple name).

```
print(len(stin)) = 4
```

→ type function is used to identify it's type as tuple.

Ex:- print(type (tuple name)) o/p:- <class 'tuple'>

Ex1 :- t1 = ("kamati", "Pravalika", "Sindhu")

t2 = (7, 9, 6)

t3 = (True, False, True)

t4 = ("Anitha", 20, True, 50, "Female")

print(t1)
print(t2)
print(t3)
print(t4)

O/p:-

('kamati', 'Pravalika', 'Sindhu')

(7, 9, 6)

(True, False, True)

('Anitha', 20, True, 50, 'Female')

#Accessing tuple items :- It can be done by using index number inside a square bracket ('[]')

p = (1, 2, 'a', 'b', 'd', 10, 11)

0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Ex

print(p[2]) => a

print(p[-5]) => a

Accessing ~~entire~~ each data element from the tuple

Ex:- k=('k', 1, 2, 'a', 'b')

```
for i in k:
    print (i)
for i in range(len(k)):
    print (k[i])
```

→ A tuple can access by using for loop with each element or for loop with index range.

Tuple Methods:- It supports 2 methods. → 1. count
2. index

Count:- This method is used to find occurance of a specific element in the tuple. no. of times of a.

Syntax:- tuplename.count(value)

Ex:- k=(6,'a','b',6,'c')
print(k.count(6))

o/p:- 2

→ If element not found it returns '0'.

Index:- It This method returns the index no of the search value.

Syntax:- tuplename.index().

→ If duplicate search values are there it returns first occurance

→ If search element is not found it raises an error

Ex:- k = (6, 'a', 'b', 6, 'c')

print(k.index(6)) o/p :- 0

List :- It used to store multiple values in a single name.

→ It can contain different data type.

→ It is collection of elements which is ordered and changable.

→ It allows duplicate values

→ It is enclosed with square bracket [].

Syntax :- name = [value1, value2 . . . .]

stin = ["Akshitha", 18, "IT-A", 'F']

→ list items are indexed it starts with zero

→ ~~List length~~

List Length :- It finds how many elements a list has

→ it can be done by using len function

Syn :- len(list name)

→ print(len(stin))

→ 4

→ type function is used to identify the it's type as list

Ex :- print(type(stin))

o/p :- <class 'list'>

Ex:- s_1 = ["pravallika", 18, "Chandur", 2003]

s_2 = ["Sindhu", 18, "Nalagonda", 2003]

print(s_1)

print(s_2)

Accessing list items:- It can be accessed by using index numbers.

l_1 = ['a', 'b', 98, 1.56]

0	1	2	3
-4	-3	-2	-1

print(l_1[0]) => a

print(l_1[-3]) => b

By Applying Slice operation we can create a sub list from the existing list.

print(l_1[1:3]) => ['b', 98]

print(l_1[0:3]) => ['a', 'b', 98]

Accessing a each element from list:- It can be done by using for loop with each element (or) with index no.

```
k=['a','b',1,2,3]
for i in k:
    print(i)
for i in range(len(k)):
    print(k[i])
```

List can be changable :-

Ex:- K=['a','b', 1, 2, 3]
print (k)
K[2]=8
print (k).

List Methods :-

1. Append :- It is used to add an element at end of the list.

name.append (value)

2. Insert :- This method is used to insert an element in a particular position.

name.insert (position, value)

3. Pop :- This method is used to delete an element from particular element

name.pop (position, value)

4. Remove :- This method is used to remove a particular value from the list

name.remove (position or value)

5. Reverse :- This method is used to reverse the list.

name.reverse( )

6. sort:- This method is used to short the list.

Ex:- K = [9, 1, 4, 2] O/p => [1, 2, 4, 9]

```
K.sort()
print(K)
```

→ A list consist of different kinds of data sort function when raise an error

Set :- It is used to store multiple values ina single line.

→ It can contain different ~~kind~~ types of data.

→ It is a collection elements which is unordered, unchangeble, unindexed.

→ It does not allow duplicate values. it is enclosed with the curly brackets '{ }'.

Syntax:- name = {value1, value2 - - - - - valuen}

Accessing a set:- A set can be access using for loop with each element.

Ex:- K = {11, 22, 33} O/p:- 11, 22, 33

```
for i in K:
    print(i)
```

Sets Supports the following methods:-

1. Add(): This method is used to add an element in to the set.

→ If element already existed it wont be added.

Syntax:- setname.add(element).

```
K = {11, 22, 33}
K.add(88)
for i in K:
    print(i)
```

pop():- This method is used to remove an element from the set.

→ It remove the first element from the set.

Syntax:- setname.pop()

remove():- This method is used to deleted a specify element from the set.

Syntax:- setname.remove(element)

Dictonary:- It is used to store the data values in key: value.

→ It is a ordered and changable collection of elements.

→ It won't allow duplicates.

→ It is enclosed in curly braces.

```
k={1:"ab", 5:"yy", 3:"zz"};
print(k)
print(k.values())
print(k.keys())
```

Accesing a dictonary:- The element in the dictonary to accessed with help of keys.

```
k={1:"ab", 5:"yy", 3:"zz"}
print(k[3])
```

```
for i in k:
    print (k[i])

k = {1:"ab", 5:"yy", 3:"zz"};
print (k[5])
```

→ It supports the following method.

1) dn.keys() - keys

2) dictonary name.values ( ) - values

3) dictonary name.clear ( ) - clear the entire content

4) dictonaryname.pop (i) - remove an element.
Key value.

Regular Expressions :- It is a sequance of characters that forms a set's pattern.

→ It is used to check whether the particular pattern avaiable in the string or not.

→ Python is having a predefined package for regular expressions

→ So to work with regular expression we have to import the regular expression.

→ To form the regular expression we using the following mega characters.

Metacharacters:-

Character	Description	Example
[ ]	A set of characters	"[a-m]"
.	Any character (expect newline character)	"he..o"
^	starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurances	"he.*o"
+	One or more occurences	"he.+o"
?	Zero (or) one occurences	"he.?o"
{}	Exactly specified no: of occurance	{2}

Regular Expression suppose the following function

→ findall ( )
→ search ( )
→ match ( )
→ split ( )
→ sub ( )

findall ( ) :-

This method is used to check a particular matter is available in the given string. It produces the output in the form of a list containing all matches.

If given pattern is not present in the string it will returns as Empty list

Syntax: re.findall("searchpattern", "stringname")

Import re

Eg: S = "Thanmai"

re.findall("Tha", a)

print(x)

O/P: ['Tha']

search():- This method is used to search a particular pattern is present in string or not. it always point out the 1st occurance of the search pattern. It produces an output in the form of a match object. If given string is not present then it will return NONE

Syntax: re.search("searchpattern", "stringname")

Eg: ~~Str~~ = "the rain in spain"

x = re.search("a.n", str)

print(x)

O/P: re.match object; span = (5, 8), match = 'ain'>

match():

→ This method is used to search a particular pattern is present in the starting of the string or not.

→ It produces an output in the form of match object

→ If the given pattern is not present then it will return NONE

Syntax: re.match("searchpattern", stringname)

Eg: import re

txt = "the rain in spain"

x = re.match("rain", txt)

print(x)

x = re.match("the", txt)

print(x)

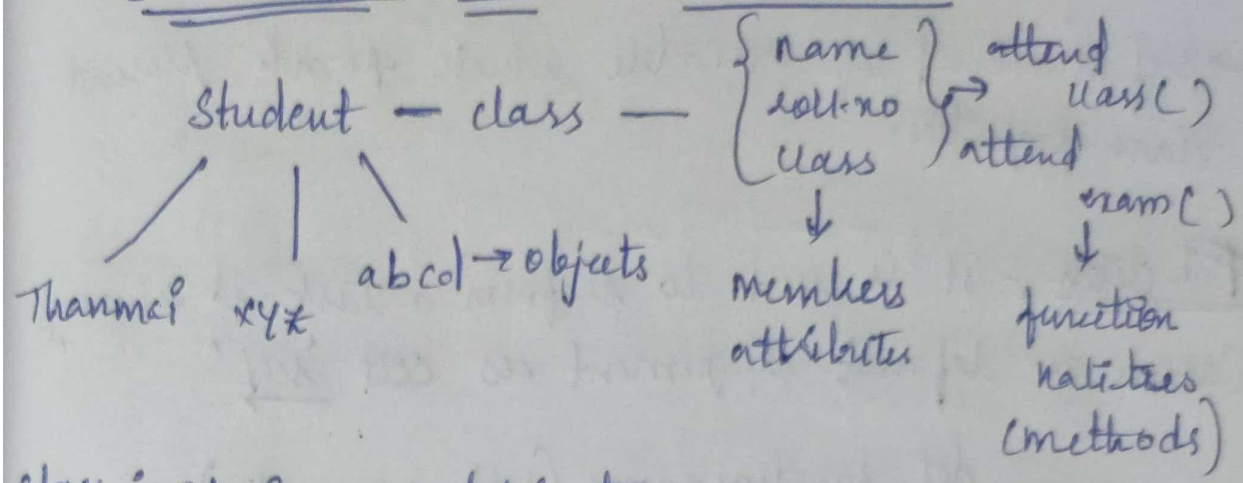
O/P: NONE

span = (0, 3)

MODULE 3

16/11/21

CLASSES AND OBJECTS



class : It is user define type

→ It ~~is a collection~~ contains of

→ It is a logical structure.

→ It is a collection of attributes, (fields), functions, constructor and etc ------

→ A class can be created in python using: **class** keyword.

Syn:

```
class classname:
    fields
    function
    statements
```

Eg:

```
class student:
    name = "Thanmai"
    branch = "IT"
    regno = 18
    def display(self):
        print(self.name, self.branch, self.regno)

s = student()
print(s.name)
s.display()
```

self.s

O/p: Thanmai

Thanmai, 21, 28

→ self is a kind of variable which points present class properties.

Function :- It is used to preform a task. it always carries one default argument as 'self'

def. functionname (self. parameter):

→ self is used as a reference variable which reference to the current class object.

→ self may be replaced with any name.

→ self is used to access the class fields

→ by default the 1st parameter in the fun definition will be treated as 'self'.

Object :- is a class variable

→ It is a having a physicall existence

→ It is used to access class field and functions

Objectname = classname (parameter) → opt

→ TO access the class members and methods the syntax is

objectname . fieldname

objectname . methodname ( ) → parameter

(optional)

Eg:
```
class student:
    def display(self)
        print("welcome to class")
s1.display().
```
o/p: welcome to class

Eg:
```
class student:
    n="Thanmai"
    b="IT"
    no=28
    def display(s):
        print(s.n, s.b, s.no)
s1.student()
s1.display().
```
o/p: Thanmai IT 28

→ 's' is treated as "self"

Eg: create a class two perform.
```
class addition:
    def display(s,x,y):
        print(x+y)
s1=addition
s1.display(10,20)
```
output = 30

* __init__( ):-

This function is called automatically when an object is created.

Syntax:-

```
def __init__(self, parameters):
                    option
```

Eg:

```
class student:
    def __init__(s)
        print("automatic")
s = student()
```

O/P: automatic

It is used to initialization fields in the object

/

2 →
```
class student():
    def marks(self):
        self.m1=20
        self.m2=20
        self.m3=19
        self.m4=19
        self.m5=20
    def display(self):
        print(((self.m1+self.m2+self.m3+self.m4+self.m5)/100)*
s1=student()
s1.marks()
s1.display()
```
o/p :- 98.0

3 →
```
class even():
    def display(self,n):
        if(n%2==0):
            print("even")
        else:
            print("odd")
e=even()
e.display(44)
```
o/p :- even.

4 →
```
class prime():
    def display(self,n):
        →n=int(input("enter"))
        if(n>1):
            for i in range(2,n):
                if(n%i==0):
                    print("not prime")
                    break
                else:
                    print("prime")
                    break
p=prime()
p.display(5)
```
o/p :- prime

```
class bank ():
    deposit = 0
    withdraw = 0
    name = 0
    def read (self):
        self.deposit = 20000
        self.balance = 40000
        self.name = "preethi"
    def display (self,balance):
        print(self.deposit, self.balance, self.name)
        if (balance >= 1000):
            print("minimum")
        else:
            print("Not minimum")
b = bank()
b.read()
b.display(40000)
```

hasattr (classname, attribute)

o/p:- 20000 40000 preethi
Minimum.

→ Palindrome or not.

```
class palin():
    def display (self,n):
        r=0
        t=n
        while(n>0):
            d = n%10
            r = r*10+d
            n = n//10
        if (t==r):
            print("Palindrome")
        else:
            print("Not Palindrome")
x = palin()
x.display(12321)
```

→ n=int(input("enter"))

o/p:- Palindrome.

Constructor and Distructor:-

Constructor:- It is a special function it is called automatically when object is created.

→ The main purpose is to initiate the data members of the objects

Syntax:- def __init__(self):
    statement

Eg:- class ct:
    def __init__(self):
        print("Constructor called automatically")
x=ct

o/p:- Constructor called automatically

→ There are two types of Constructors i, default Constructor
ii, parametrised Constructor

default Constructor:- A constructor without parameters is called as default Constructors

Syntax:- def __init__(self):
    statements

→ default Constructors can be called when an object is created without passing the parameters.

object name - class name.

Parameterised Constructor:- A Constructor with parameters is called as parameterised Constructor

→ It can be called when an object is created by passing the parameters

object name : class name (parameter)

```
class ct:
    def __init__(self,x)
        print("par Con called automatically")
c1=ct(33)
```

Note:- Constructor overloading is not supported by py because it is not having specified data types

→ If a class Contain more no. of Constructor Last Constructor over riding on all previous Constructor. It is going to get executed.

Ex:-
```
class ct:
    def __init__(self):
        print(" Con called automatically")
    def __init__(self):
        print(" Con called automatically")
c1=ct()
```
o/p:- Con called automatically

Destructor:- It is a special function.
It is called automatically when an objed is destroyed from the memory.

```
def __del__(self):
    statements
```

↪ To call the destructor externally use delete statement

Syntax

```
del objectname.
```

```
class ct:
    def __del__(self):
        print("des called")
c1=ct()
del c1
class ct:
    def __del__(self):
        print("des called")
    def test(self):
        c2=ct()
        c3=ct()
        c4=ct()
c1=ct()
c1.test()
```

o/p:-
des called
des called
des called

Access specifiers (or) Access modifiers:-

↪ Access specifiers are used to restrict the acess of class members & functions of the class.

→ In python we are using "__" symbol to create access specifiers

→ There are 3 access specifiers i, public
ii, protected
iii. private

Public:-

→ Public access specifiers allows the data and functions to access outside of the class

→ This is a default access specifiers in python

→ To make a data member or function as public don't add any

underscore symbol before name.

Syntax: data member:

name = value

function:

def functionname (self):

Ex:- class pubinh:

a=33

def show(self):

print("public data =", self.a)

pd. pubinh()

pd.show()

Protecter:-

→ protected access specifier doesn't allow the data at function to access outside of the class.

→ Protected data is 'inhertable' to the derived class.

→ To make a function or data member as protected, add a underscore symbol before the name.

Syntax:- data member:

-name =value

function:

def -functionname(self):

Ex:- class prtinh:

-a=33

def -show(self):

print("protected = ", self.-a)

pd = prtinh()

pd.-show()

Private:

→ Private access specifier does not allow the function & data to access outside of the class

→ It is not inheritiable

→ To make a function or data member as a private add two 'under score "symbol before the name.

Syntax:- data member:

```
__name = value
def __function(self):
```

Ex:-
```
class priinh:
    __a=33
    def __show(self):
        print("private =", self.__a)
    def priacc(self):
        self.__show()
pd.priinh()
pd.priacc()
```

Built in functions to check, get, set, delete class artibutes.

i) Checking the artibute:- To implement this we are using 'hasattr()' function

→ This function will return boolean values either 'True' or 'False'.

→ If the artibute is existed then return True otherwise False.

Syntax:- hasattr(classname (or) objectname, 'artibute name')

ii) Get the artibute:- It is implement by using "getattr()" function

Syntax:- getattr(classname or o.n, 'artibutename')

→ This function will return the value holding by the artibute. or it will rises an error if the artibute is not avaiable.

iii, Set artibute:- It is implemented by using "setattr()".

Syntax:- setattr(classname or obj.na, 'artibutename', value)

iv, deleting artibute:- It is implemented by using "delattr()".

→ Class have right to delete the atribute.

→ object doesn't have right to delete the atribute. (practical).

Syntax:- delattr(classname, 'artibutename')

→ If the artibute is not present it rises the error.

→ It return the NONE if the artibute is present.

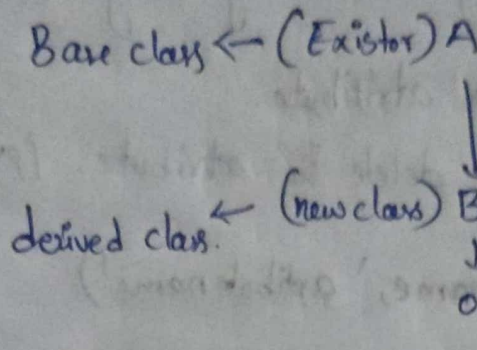
Ex:-

```
class K:
    a=44
    b="pravalika"
    c=8.9
    def dis(self):
        print(self.a, self.b, self.c)
K=K()
K1.dis()
print(hasattr(K,'a'))
print(hasattr(K,'u'))
print(getattr(K,'a'))
setattr(K,'a',88)
print(hasattr(K,'a'))
print(delattr(K,'b'))
```

*** Imp Long.

Inheritance:- The process of deriving a new class from the existing class is called inheritance.

↪



→ Here existing class is called Base class / parent class / super class

→ Here new class is called as derived class / child class / sub class.

Derived class syntax:-

```
class derivedclassname (Baseclassname):
    Attributes
    functions
```

↪ **Eg**:-

```
class A:
    def f1(self):
        print("Base class")

class B(A):
    def f2(self):
        print("Derived class")

b1 = B()
b1.f1()
b1.f2().
```

o/p:- Base class
Derived class

Types of Inheritance. i, Single inheritance,
ii, Multiple inheritance.
iii, Multi level inheritance
iv, herarical inheritance. → A(base) → B, C (derived)

Single Inheritance :- (base) A → B (derived)
f_1 $f_2 + (f_1)$

→ The process of deriving a class from only one base class is called single inheritance.

→ Syntax :- class baseclass:
    statements

class derivedclass (baseclass):
    statements

i, Ex :- class A:
    def f1(s):
        print ("base class A")

class B(A):
    def f2(s):
        print ("derived class B")

b1 = B()
b1.f1()
b1.f2()

Multiple Inheritance :- The process of deriving a class from more than one base class is called multiple inheritance.

Syntax :- class baseclass 1:
    ⋮
class base class 2:
    ⋮
class derived (baseclass 1, base class 2):
    ⋮

(base classes)
A B
\ /
C
(derived classes).

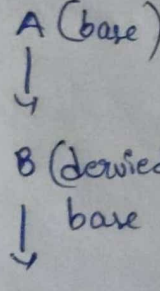
Ex:-

```
class A:
    def f1(s):
        print("base class A")
class B:
    def f2(s):
        print("base class B")
class C(A,B):
    def f3(s):
        print("derived class C")

C1=C()
C1.f1()
C1.f2()
C1.f3()
```

iii, Multilevel:- The process of deriving a class from another derived class is called Multilevel inheritance.

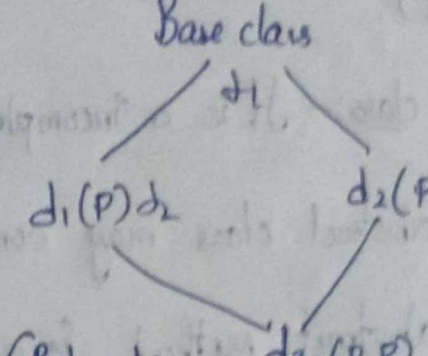
Ex:-




```
class A:
    def f1(s):
        print("base class A")
class B(A):
    def f2(s):
        print("base & derived B")
class C(B):
    def f3(s):
        print("derived class C")

C1=C()
C1.f1()
C1.f2()
C1.f3()
```

Multipath inheritance:- The process of deriving a class from more than one derived classes which are derived from the same class is called Multipath inheritance.




```
class A:
    def d1(6):
        print("base class")
class B(A):
    def d2(S):
        print("derived class1")
class C(A):
    def d3(S):
        print("derived class 2")
class D(B,C):
    def du(S):
        print("Sub derived class")
D1 = D()
D1.d1()
D1.d2()
D1.d3()
D1.du()
```

Super():- This statement is used to acts as super class members & functions.

Syntax:- super().function name()

Ex:-
```
Class A:
    def d1(S):
        print("base class")
Class B(A):
    def d2(S):
```

```
        print("Derived class")
        super().fn()

B1 = B()

B.fn2().
```

imp.

Abstract class:- It is a incomplete class.

↪ An abstract class may contain one or more abstract method

↪ An abstract method is a method without defination.

↪ We can't create an object for an abstrad class.

↪ In python abstraction is using 'abc' package.

```
from abc import ABC, abstractmethod
```

→ Abstraction can be implemented by using inheritance.

→ Here derived class method over ridded on base class abstract method.

↪ It means we are providing the defination for abstract method in the derived class.

→ If we are trying to create an object for abstract class it raises an error.

```
from abc import ABC, abstractmethod
class A(ABC):
    @abstractmethod
    def fn(s):
        pass
    def fn2(s):
        print("derived")
```

```
class B(A):
    def f3(s):
        print("redefined")
b1 = B()
b1.f3()
b1.f2()
```

o/p :- redefined
defined -

Polymorphism

→ Method-2 for making an abstract class.
→ It is achive by raising a non implemented error.

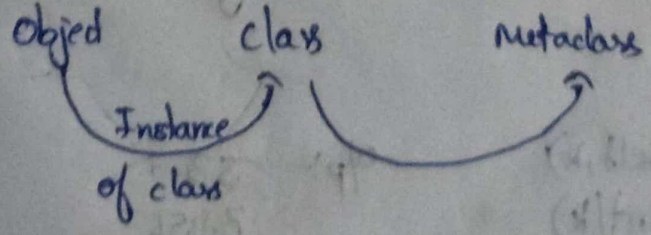
```
class A:
    def f1(self):
        raise Not implemented error
    def f2(self):
        print("Defined")

class B(A):
    def f1(self):
        print("Defined in derived class")

a1 = B()
a1.f1()
a1.f2()
```

o/p :- Defined in derived class
Defined.

Meta Class :- A meta class is the class of a class.



Meta class define how a class behave.

→ It is acted as a class factory.

→ To create a own meta class we have to redefined

1) __init__()

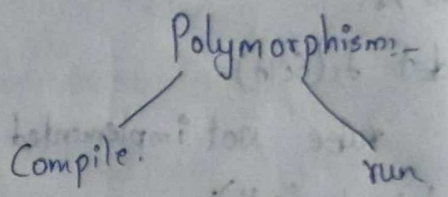
2) __new__()

→ even meta class allows us to defined our own method.

Ex:- class A:

```
    def __init__(self):
        print("defined")
print(type(A))
a1=A()
print(type(a))
```

Polymorphism:-



=> Poly - many morphism = forms

When one thing is perform multiple functionalities that situation is called as polymorphism.

There are two types of polymorphism 1) Compile time
2) Run time

Compile time Polymorphism:-

function overloading:- It is a function which can perform different tasks types of tasks known as function overloading.

Ex:- class A:

```
    def d1(self,k)
        print(k)
a1=A()
a1.d1(33)
a1.d1(3.456)
```

o/p:- 33
3.456
Ravalika.

a1.fn("Pravalika")

operator overloading:-

→ The process of providing the new meaning to the existing operators is called as operator overloading.

→ It allows program to provide it's own defination for an operator

→ It can be achieve by using specifiy predefined methods in the python environment. (operator function)

```
class A:
    a=0
    def read(self):
        self.a=int(input("enter a no"))
    def dis(self):
        print(self.a)
    def __add__(ob1,ob2):
        return(ob1.a+ob2.a)

a1=A()
a2=A()
a1.read()
a2.read()
a1.dis()
a2.dis()
print(a1+a2)
```

O/p:- 33
44
77

Operator	function name
+	__add__
-	__sub__
*	__mul__
/	__truediv__
* *	__pow__

%	--mod--
>>	--rshift--
&	--and--
|	--or--
^	--xor--
~	--invert--
<<	--lshift--
>	--gt--
<	--lt--
>=	--ge--
+=	--iadd--
-=	--isub--
*=	--imul--
/=	--idiv--
==	--eq--
!=	--ne--
&=	--iand--
|=	--ior--
^=	--ixor--

In python for each operator a predefined operation function is availabl
To overload an operator we have to use that respective predefined operator functions

In the before page program a_1+a_2 will call --add--() function

Overloading a multiplication operator:-

```
class A:
    a=0
    def read(self):
```

```
    self.a = int(input("enter no"))
def dis(self):
    print(self.a)
def __mul__(ob1, ob2):
    return(ob1.a * ob2.a)
a1 = A()
a2 = A()
a1.read()
a2.read()
a1.dis()
a2.dis()
print("product of objects", a1 * a2)
```

O/p:- enter no 12
enter no 13
12
13
product of objects 156

Overloading relational operators:- (<, >, <=, >=, ==, !=)

```
class A:
    a=0
    def read(self):
        self.a = int(input("enter no"))
    def dis(self):
        print(self.a)
    def __gt__(ob1, ob2):
        return(ob1.a > ob2.a)
a1 = A()
a2 = A()
a1.read()
a2.read()
a1.dis()
a2.dis()
if (a1 > a2):
    print("obj1 > obj2")
else:
    print("obj2 > obj1")
```

O/p:- enter no 12
enter no 13
obj2 > obj1

Advantages of Operator overloading:-

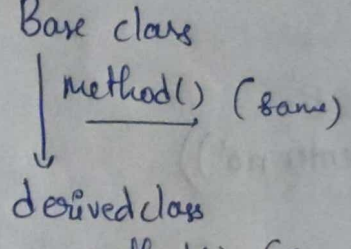
1. → We can redefine the meaning of the operator.
2. In operator overloading we can use the same operator notations like arthimetic operators (obj1+obj2) => Ex.
3. It is used mainly in scientific computing programs.

Method Overridding (function Overridding):-

→ It belongs to inheritance.

→ The When base class & derived class are having the same method derived class method overridding the base class method.

→ This process is called method overridding.




```
class A:
    def fi(self, x):
        print ("base class
class B(A):
    def fi(self):
        print ("Derived clas
b1 = B()
b1.fi()
```

o/p:-

longest-

Difference blw method overridding & overloading

Method Overloading	Overridding
→ Compile time phe polymorphism	→ Run time polymorphism
→ Here method name must be same with different list of parameters.	→ Here method name & param must be same.
→ Overloading doesn't require inheritance	→ It require inheritance
→ It performs within the class	→ It perform in derived class

- It is used to provide additional behaviour to the existing method name.

→ It is used to redefine the existing behaviour

Ex:-

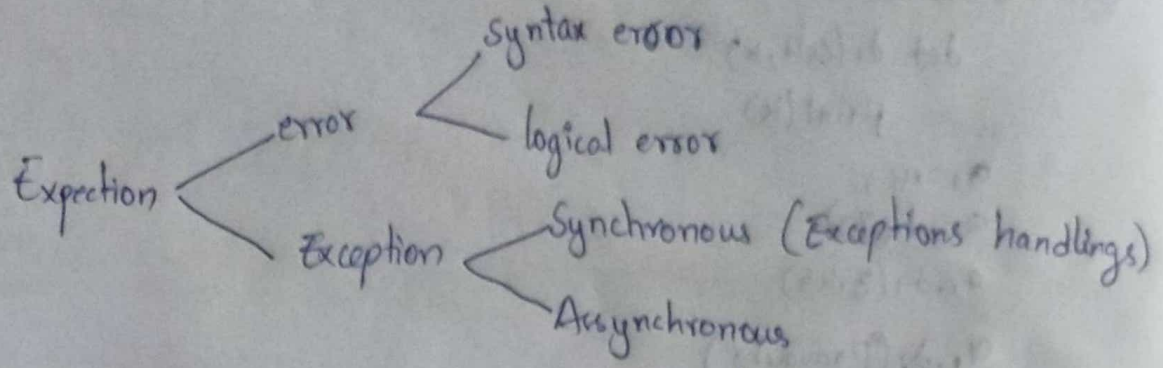
```
class A:
    def fn(self,k)
        print(k)

a1=A()
a1.fn(33)
a1.fn(3.45)
a1.fn("Pravalika")
```

O/P:- 33
3.45
pravalika.

Exception Handling:-

→ Basically There are two types of errors 1) errors
2) exceptions



Exception:-

→ An exception is a run time error. It occurs during execution of the program.

→ It causes abnormal termination of the program

→ Abnormal termination causes loss of data.

→ To avoid the situation exception should be handled.

Exception handling:-

→ It is a mechanism or process to handle the exception without abnormal termination of the program.

→ It can be handled with help of try & except block.

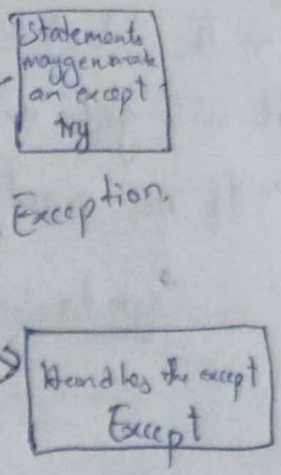
→ Try block:-

→ It is collection one (or) more statements which may generate an exception.

→ Except block:-

→ This block catches the exception raised in the try block &

handles the exception.



Syntax:- try:
    statements
  except exception name:
    statements

Ex:- ZeroDivisionError → dividing by zero.
IndexError → index out of range

Ex:- Write a program to describe about exception handling mechanism of

```
a=int(input("enter no:"))
b=int(input("enter no:"))
try:
    print(a/b)
except ZeroDivisionError:
    print("Dividing by zero not possible")
print("end of program")
```

o/p:- enter no: 40
enter no: 5
8
end of program.

Ex 2:-
```
a=[1,2,3]
try:
    print(a[4])
except IndexError:
    print("index out of range")
print("end of the program")
```

→ Multiple Except blocks:- Python allows u to have multiple except block for a single try block.

→ The block which matches with the exception generated it will get executed.

→ If no exact match found it raises exception

Syntax:-

```
try:
    statements
except exception name1:
    statements
except exception name2:
    statements
    .
    .
except exception nameN:
    statements
else:
    statements
```

→ when no exception raises then else block will execute (optional)

Eg:- a=[3,0,5] generated

```
try:
    print(a[2]/a[1])
except IndexError:
    print("Index out of range")
except ZeroDivisionError:
    print("dividing by zero not possible")
else:
    print("no exception")
```

Multiple exceptions in a single block:-

→ python allows to work with multiple exceptions under a single except block

→ It can be done by defining the list of exceptions as a tuple under except block.

Syntax:- try:
    statements
except (exception name 1, exception name 2):
    statements

→ Here the draw back is getting a generalised solution for exception

Ex:- a=[0,2,3,4]
try:
  print(a[8]/a[0])
except (ZeroDivisionError, IndexError):
  print("diving by 0 not possible or index out of range)

O/p:- diving by 0 not possible or index out of range

Out of Syllabus:-
→ Except block without Exception (Generic Except Block):-
→ This except block should be added after adding the all except blocks.
→ It is used to handle any type of exception.

Ex:- try:
    statements
except exceptionname:
    statements
except expection name:
    statements
except:
    statements

Ex:- a=[3,0,5]
try:
  print(a[2]/a[1])
except IndexError:
  print("index out of range")

```
except:
    print("Exception raised")
```

Finally :-

→ This statement is associated with try & except block

→ This statement will executed always (even exception raised or not raised)

→ It is mainly used to perform a task at the end of exception handling.

Ex
```
a=33
b=9
try:
    print(a/b)
except ZeroDivisionError:
    print("div by 0 not pos")
finally:
    print("always executed")
```

Numpy :-

info 18

UNIT-5

Numpy:-

→ numpy stands for numerical python.

→ numpy is a python library used for working with arrays.

→ It also has functions for working in domain areas like linear algebra, fourier transformation & Matrices.

→ Numpy provides the concept of array of objects which is of every faster than list concept.

→ It supports continoues memory allocation.

→ To work with numpy we have to install numpy in python environment.

Syntax:- For installing Numpy:-

pip install numpy

→ Once after successful installation of numpy import it into your application by the following statement.

→ ~~Imp~~ Syntax import numpy

→ Working With arrays:-

→ Step 1:- Creat a numpy ~~and the~~ nd array object.

→ It is used to work with arrays.

→ We can create the nd array object by using array function.

Syntax:- Objectname = numpy.array (elements)
List (or) tuple

(or)

Arrayname.

→ It supports 3 types of arrays:- 1) 0 dimensional. (individu
2) 1 dimensional.
3) 2 dimensional
4) Multidimensional.

0 dimensional:- It is a call single element.

Ex:- import numpy as np
arr = np.array(33)
print(type(arr)) ⟶ nd array object
print(arr.ndim) ⟶ 0

1 dimensional :- It is a collection of elements.

→ It is representation of one row with multiple columns.
→ A 1d array is a collection of 0d elements.

Ex:- import numpy as np
arr = np.array([22,33,44])
print(type(arr)) ⟶ nd array object
print(arr.ndim) ⟶ 1
print(arr) = [22,33,44]
Accessing individual
print(arr[2])

Syntax:- Object name = numpy.array([elements])
Array name
(0D (elements)

Displaying

Accessing individual element from the array:-

for i in range(0,5):
print(arr[i])

Ex: `import numpy`

```
arr = nump
```

→ Write a program code to find the sum of all elements in the array and display it.

```
import numpy as np
arr = np.array([1,2,3,4])
s=0
for i in range(0,4):
    s = s+arr[i]
print(s)
```

replace highest in lowest

2nd biggest

2D- Array:- It is represented in terms of rows & columns.

→ It is also called as Matrix.

Ex:- import numpy as p

```
mat = p.array([[1,2,3],[4,5,6]])
print(mat)
```

output:- [[1, 2, 3]
[4, 5, 6]]

```
for i in mat:
    print(i)
```

```
for i in range(0,2):
    for j in range(0,3):
        print(mat[i][j], end=" ")
    print(" ")
```

1 2 3 4
5 6 7

Properties and functions supported by array:

Properties:

i; ndim:- This property is used to find the dimensions of array.

Syntax:- arrayname.ndim

ii; Shape:- It is used to find the row size and column size of an array.

Syntax:- arrayname.shape

ii, Size:- It is used to find the number of elements in the array

syntax:- arrayname.size.

iii, dtype:- It is used to find the data type of the array.

Syntax:- arrayname.dtype.

Eg:-
```
import numpy as kk
arr = kk.array([[5,6,7],[4,5,6]])
print(arr)
```

o/p:-
```
[[5, 6, 7]
 [4,5,6]]
```

```
print(arr.ndim)      o/p:- 2
print(arr.shape)     (2,3)
print(arr.size)      6
print(arr.dtype)     int 32
```

Slicing arrays:-

This operation is used to extract the sub part from the array.

Syntax:- arrayname [start:end:step]

→ If we don't pass start it is considered as 0.
→ If we don't pass end it is considered as length of the array
→ By default step count treated as 1.

Ex:- arr2 = kk.array([4,5,6,7,8,9])

print(arr2[1:4])

print(arr2[:5])

print(arr2[0:6:2])

O/p:- [5,6,7]

[4,5,6,7,8]

[4,6,8]

2-D array slicing:-

Syntax:- arrayname([rowslicing, columnslicing])

Eg:- arr2 = kk.array([4,5,6],[7,8,9],[2,3,4])

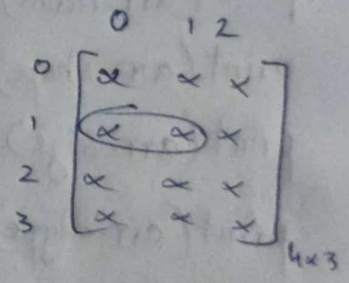
print(arr2[1:3,1:3])

O/p:- [[8,9]

[3,4]]

print(arr2[0:3,2])

[6 9 4]



arr(1,0:2)

row column

Data Types in Numpy:-

→ Datatypes are used to specify the type of data

→ Numpy supporting the following data types.

i - integer	M - date time
b - boolean	O - object
u - unsigned integer	S - string
f - float	U - Unicode string
c - Complex float	V → fixed chunk of memory for other type (void)
m - time delta.	

Creating a array with defined data type:-

In python we can define an array by specifying it's data type

It can be done by using d-type optional argument.

Eg:- import numpy as kk

```
arr = kk.array([5,6,7.8,33]), dtype = "i")
print(arr)
print(arr.dtype)
```

o/p: [5 6 7 33]

int 32

Converting data type on existing arrays:-

→ It is possible by using astype().

→ It creates a copy of the array with the specified type.

```
arr = np.array([1.1, 2.1, 3.1])
new arr = arr.astype("i")
```

Copy():-

This function is used to create the another copy of the array.

Eg:-
```
import numpy as kk
arr=kk.array ([5,6,7,33])
print (arr)
arr1=arr.copy()
print (arr1)
```

O/p:- ~~[5,6,7,~~
[5 6 7 33]
[5 6 7 33]

Reshape():-

This function is used to create a new array with dimensions from the existing array

Eg:-
```
import numpy as kk
arr=kk.array ([5,6,7,33,66,77])
print (arr)
arr1=arr.reshape (2,3)
print (arr1)
```

O/p:- [5 6 7 33 66 77]
[[5 6 7]
[33 66 77]]

Joining arrays:- It is done by using Concatenate()

Syntax:- numpy.concatenate (array1, array2.. array))

Eg:-
```
import numpy as kk
arr1 =kk.array ([5,6,7,33,66,77])
arr2 = kk.array ([51,64,74,3])
arr3 = kk.concatenate ((arr1,arr2))
print (arr3)
```

Splitting array:- It is done by using array-split() method

It is used to divide an array into sub arrays.

Eg:-
```
arr1 = kk.array ([5,6,7,33,66,77])
arr2 = kk.array-split (arr1,5)
print (arr2)
```

Searching Arrays :-

→ It can done by using where() function.

→ This function returns the set of indices which gets exact match

Syntax:- np.where (condition, output-if-true , output-if-false)

Eg:- import numpy as kk

arr = kk.array [5,6,7,8,9,77,33]

Sorting Arrays:-

→ sort() is used to perform the sort operation on the array

→ It display the elements by default in the ascending order.

Eg:- import numpy as kk

arr = kk.array ([5,66,17,43,6,77,33])

print (kk.sort(arr))

print(-kk.sort (-arr))

-np sort (-arr)

Panda's:-

→ It is a python library used for working with data set.

Ex:- Excel data, database data,

→ It has a functionality's like analysing, cleaning, exploring and manipulating the data.

→ The name Panda reference to panel data & python data analysis.

→ It was created by McKinney in 2008.

Advantages:-

1) It is used to analyze the big data and may conclusion

2) It is used to clean the data sets and make them useful.

3) It plays the imp role in data science & data analysis.

Installation:-

→ It can be done by following Command.

pip install pandas

→ Once it installed succesfully we can using by importing the package panda's.

Syntax:- import pandas

→ We can create an alternative need by using "as" keyword in import statment

Syntax:- import pandas as name.

Ex:- import pandas as pp.

Pandas → Package ← Excel data → rows x cols → dataframe

arr = np.array([[1,2,3], [4,5,6], [7,8,9]])

r1 r2 r3

DataFrame:-

→ It is a 2Dimensional data structure.

→ It represents the data in the form of rows x cols in a tabular for

→ Features of Data frame:-

1) Potentially columns are of different types.

2) Size - mutable

3) Labeled axes (rows & columns)

4) Can perform Arithemetic operations on rows & columns.

Ex:- . (rows → sn1–sn5; S.no, name, age → column.)

	S.no	name	age
sn1	1	Pravalika.	19
sn2	2	Harika	19
sn3	3	Deepika	19
sn4	4	Akshitha	19
sn5	5	Mahanya	19

Creation of Data frame

→ It can be created by using a method dataframe.

Syntax:- pandas.DataFrame (data, index (rows), columns, dtype, copy)

Ex 1:-

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

O/p:-

	0
0	1
1	2
2	3
3	4
4	5

Ex 2:-

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

O/p:-

	sno
0	1
1	2
2	3
3	4
4	5

→ To read the data from excel. we are using (real_excel) method.

→ To read csv file we are using (real_csv) method.

→ To describe the data we are using the following methods.

1) head (show first 5 lines of data)
2) tail (show last 5 lines of data)
3) Sample (Number) (show random data)

Data filtering: (slicing the data)

→ loc () and iloc () methods are used in slicing data from

the pandas.

loc() :- select by label.

iloc() :- select by positions.

Groupby() :- This method is used to spliting the dataset as per the categories.

Data Clearing :-

→ Removing empty cells by dropna. method

→ Fill the empty cells by fillna method.

Data Visualization :-

→ It can be done by using package matplotlib.

Installation :-

pip install matplotlib