ANNAMACHARYA UNIVERSITY

SUBJECT NAME:- THE JOY OF COMPUTING USING PYTHON

SUBJECT CODE:-24ACSE23T

Prepared
By

P.MABJAN, MCA (Assistant Professor)

Dept. Of AI&DS

Introduction to Phyton programming

Introduction to Phyton:-

Phyton history:- The implementation of phyton was started in Dec 1989 by Guido van Rossum at CWI in Netherland. but offical phyton was made available to public in 1991. the offical date of Phyton is Feb 20 1991.

* In 1994 phyton-1.0 was released with new-features like Lambda, Map, filter, reduce.
* Phyton-2.0 added new features such as list comprehsion. Garbage collection system. on Dec 3, 2008 it
* on Dec-3, 2008 phyton-3.0 was released. It was designed to rectify the fundamental flaw of the language.
* There is a fact behind choosing the name phyton. "Guido van Rossum" was reading the script of a popular BBC comdey series " Monty's python's flying circus". It was late on 1970's
* Van Rossum wanted to select a name which is unique, sort and little bit mysterious. so he decided to select naming phyton after the monty Python's flying circus for their newly created programming language.
* Guido developed phyton language by taking almost all programming features from different languages.

1) Functional programming features from C.
2) Object oriented programming features from C++
3) scripting language features from perl and shell script
4) Modular programming features from Modula-3
5) Most of syntax in phyton derived from C and ABC languages

Where we can use phyton:-

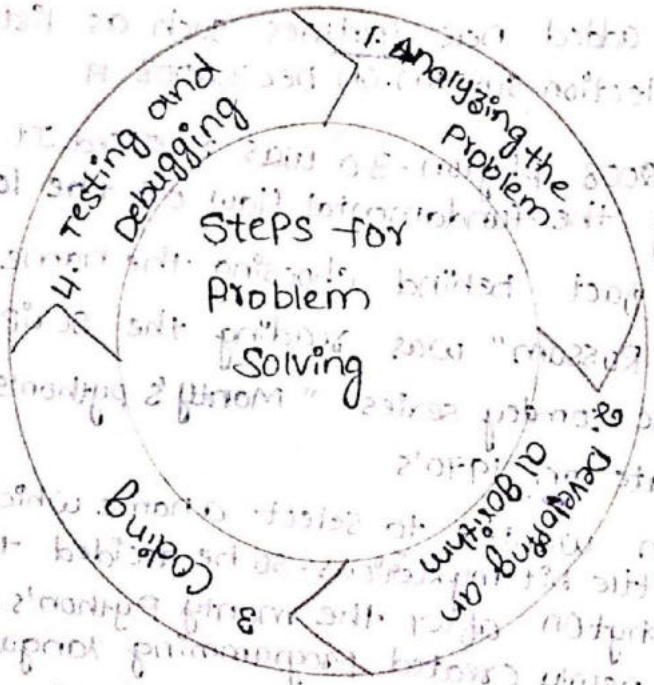
* we can use phyton everywhere. the most common important application areas are.

1) For developing desktop applications
2) For developing web applications
3) For developing database applications
4) For developing Games.

data analysis applications
6) For machine learning
7) For IoT
8) For Artificial intelligence applications

Computational problem solving:-
The process of computational problem solving involves understanding the problem, designing a solution and writing the solve.

Steps for problem solving



1. Analyzing the problem :- It is important to clearly understand the problem before we begin to find the solution for it. if we are not clear as to what is to be solved, we may end up developing a program which may not solve our porpous thus. we need to read and analyze the problem statement carefully in order to list the principle components after problem or decesied the core functionalities that our solver should have. by analyzing a problem we would be able to figure out what are inputs that our problem should accept and the ~~outputs~~ outputs that it should produce

2. Desining an algorithm:- It is essential to device a solution before writting a program code for a given problem. The solve is represented in natural language it is called an algorithm. we can image an algorithm like a very well return recipe for a dish, with clearly defin

steps that it followed, one will endup preparing the dish.

* algorithm: In our day-to day life we perform activites by following certain sequence of step. example of activities include getting ready for school, making breakfast, riding bicycle wearing Id card and soon. to complete each activity we follow sequence of steps. suppose following or the steps require for an activity riding a bicycle.

1. Remove the bicycle from stand.
2. Sit on the seat of the bicycle. ~~and~~
3. Start Peddling
4. use breaks when every need.
5. Stop the bicycle on reaching the desination.

3) Coding :- after finalizing the algorithm we need to convert the algorithm into the format which can we understood by the computer. to generate the designed solver different high level programming language can be used for writting a program. It is equally important to record the details of the coding processuor followed and document the solven.

4) Testing and debugging :-

The program created should be tested on varies parameters. the program should met the requirments of the user. it must respond with in the expected time. it should generate collect output for all possible inputs. in the presence of synthetical error no output will be obtained. In case the output generated is interepter, then the program should be checked for logical error if any.

software industry, follows standard testing methods like unit compound testing, integration testing, system testing, and exptend testing. while developing complex applications.

This is to ensure that the software mets all the buisness and techinally requirments and works as expected. the errors found in the testing phases or Deggugor recting and the program is again tested. this continous still all the errors are removed from the program. this is about computational Programming

what is Phyton?

Phyton is simple, dynamic, general purpose, intepter and high level programming and also object oriented Programming language. Phyton is much easier than other programming languages and helps you create beautiful applications with less effect and much more easy

Feature of Phyton:-

1). Simple and easy to learn
2). free usage and open source.
3) high level programming language.
4. Plateform independent.
5. Portability
6. Dynamically typed
7. Both ~~Pre~~ and Object oriented
8. Interpter
9. exstansble
10. ~~emvededd~~ Embedded
11. extensive libary.

1) Simple and easy to learn:

* Phyton is simple programming language when we read Phyton we can feel like reading english statements
* These syntaxs are very simple and only 30+ keywords are available
* when compare to other languages we can read write less number of lines hence phyton is more readibity and simplicity
* we can reduse development and cost of the project

2) freeware and open source

* we can use phyton software without any license and it is freeware.
* It is source code is open so that we can customised based on our requirment.
  ex: jython is customised version. Phyton to work with java applications

3) High level programming language:

* phyton is high level programming language and hence it is programming friendly language.
* Beging a programmer we are not required to concrate low level activities like memory managment activities, security

4) Plate-form independent:

* once we write a phyton program it can run any plat-form with out brief writting once again
* Internal PVM is responbonsile to convert into machine understandi form
  PVM- Phyton virtualmachine
* Phyton

5) * Portability:

* Phyton program are portable we can migrate from Plat-form to another plat-form very easily Phyton program will provide same result on any plate form

6) Dynamically typed language.

* In phyton we are not required to declare type for variables whenever we are assigning a value based on the value type will be allocated automatically hence phyton is consider as Dynamically typed language.
* But Java, C are statically typed languages we have to

Provided type at beginning only.

* these dynamic typing nature will provided more flexibility to the programmer

* Both procedure oriented and object oriented.
Phyton language supports both procedure oriented and objected oriented futhers hence we can get benefits both like security and reusbility.

* Interpter:

* we are not require to compile phyton programm explicit internaly phyton interpteri we taken care of that compilation.

* If compilation fails interpter raised syntax errors once compilation sucess then PVM is responsible to execute

* Extansable:-

* we can use other language program in phyton.

* The main advantage of this approach are we can use already existing legacy non-phyton code

* we can improve performance of the applications

* Embedded:

* we can use phyton programs, in other language programs that is we can embedded any phyton programs any where.

* Extensive library:-

* Phyton has a rich enghi in built library. begin a programmer we can use this libary directly and we are not responsible to implement the functionality etc. are not responsible to implement the functionality etc.

* Limatation of Phyton:-

* performance vise not upto the mark because it is interpter language.

* Not using for mobile applications.

* flavours of Phyton:-

* C. Phyton

Description:- It is a standard flavour of phyton it can used to in C-language application.

Jython(or) J-Phyton:- It is used for java applications

Iron-Phyton :- It is for C-dotnet appli. Platform

PyPy :- The main advantage of PyPy is Performance will be improved because JIT Compiler is available inside PVM

Ruby python :- It is used for Ruby Platform

Ana konda python :- It is special ~~desised for~~ designed having large Volume of data proccesing.

Python versions :-

Advantages and disadvantages of Phytons :-

Advantages :-

* Open source
* easy to learn and explore.
* thirdpaarty modules can be easily integrated
* It is High level programming language. and Object oriented Programming language
* It is interactive and portable.
* applications can be run on any Platform.
* It is dynamically typed language
* It has great only support it is user friendly data structure.
* It has extensive suppor libraries.
* It is interpter language.
* Phyton Provides data base connectivity
* It improves programmer productivity

Disadvantages :-

* It cannot be used for mobile application development
* It has limations with data base excess.
* It throws runtime isuee the passes the isuee for of the programmer.
* It consumes more memory because of dyramically typed language.
* It's speed is low.
* Need more maintanece of applications and code.

Identifiers :-

* Identifiers are the names given to any variable, function, Class, list, method etc for their identifaction. Phyton is a case sensitivity language and it has some rules and regulation to name an identifier. name in phyton program is called identifier. it can be class name, function name (or) variable name. here are some rules to name an identifier.
* In phyton we cannot use $.

* Alphabet symbols either lower (or) upper case.
* The identifier starts with underscore(_) then it indicates it is private
* Identifier should not starts with digits. we cannot use reserved keywords as identifiers

  Eg:- def = 10

* There is no length limit for phyton identifiers but not recommended to use length identifiers.
* ($)Dollar is not allowed in phyton.

Rules to define identifiers in Phyton:

* The only allowed characters allowed in phyton are
  1) Alphabet symbols either lowercase or uppercase
  2) Digits 0 to 9
  3) Underscore symbol
  4) By mistake if we are using other symbol like Dollar($) then it will get syntax error.
     ex:- - SPEED 123 Speed limit
  5) identifiers should not starts with digits
     ex:- 123 total = 10 X
          total 123 = 20 ✓
  6) identifiers are case sensivity of course phyton language is case sensitive language.

ex:- total = 10
     TOTAL = 20
print(total) - 10

Phython Variables:-

* ~~Phython is a~~
* Variable is a name that is used to refer to memory locadio Phyton variable is also known as an identifier and it is used to hold values and in Phyton we don't need specify the type of variables.
* we don't need to specify the type of variables because Phython is a inter language smart enough to get variable type
* Variable can be groups of both the letters and digits, but they have begin with a letter (or) an under score. it is recommend to use lower case letters for the variable name.

Rules for greating variables in Phyton?

* Variables are the example of identifiers, an identifier is used to identify literals used in the Program. the rules to name an Identifier are given below.
* The first character of the variable must be an alphabet or (-)
* All the characters except the first character maybe an alphabet of lower case (a-z), Upper case (A-Z), Underscore, digits (0-9). identifier name must not contain an white spaces or special characters (!, @, #, !, %.) @ ① @gmail.com
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensivity for example:- Myname, MyName are not same.
* example of valid identifiers: a123, _n, n_9 etc
* example of invalid identifiers: 1a, n-1-4, n 9, etc.

* Declaring variable and assing values:-

Variable names:-

Variable names can be an ~~variable~~ name can have upper case, lower case (A to z, a to z) digits (0 to 9) (-). consdier the following example of vallide variable names.

exprogram:-

```
name = "Hanvika"
age = 20
Marks = 80.5
print ("name")
print ("age")
print (marks)
```

```
name = "Hanvika"
age = 20
Marks = 80.5
print (name)
print (age)
print (marks)
```

Output:-
Hanvika
20
80.5

~~ass~~ multiple assignments.

Phyton allows to assign a value to multiple variables in a single statement which is also knows as multiple assignments. we can apply multiple assignments in two ways either by assign a single value to multiple variables (or) assign multiple value to multiple variables.

Program:-

```
x = y = z = 20
print (x)
Print(y)
print(z)
```

Output:-
20
20
20

accessing multiple values to multiple variables:

ex Program:-

```
1. a,b,c = 5,6,7
2. print(a)
3. print(b)
4. print(c)
```

Output:- 5
6
7

Phyton variable types

They are two types of variables types

1) local variable
2) Global variable

local variable:- local variables are the variables that declaerted inside the function and have ~~spoke in~~ scope with in the function.

example Program:-

```
1. def add();
   a=20
   b=30
   c=a+b
5. print("The sum is
6. add()
```

```
1. def add():
2.     a=20
3.     b=30
4.     c=a+b
5.     print("The Sum is",c)
6. add()
```

Output:-

The sum is 50

Global variable:-

Global variable can be used throughout the program and its scope is in the entire program. we can use global variable inside and outside of function. a variable declaerted outside the function is the global variable by defilate. default. Python provides the global keyword to use global variable inside the function. we don't use global keyword the function treats it as a local variable.

ex program:-

```
1. X=100
2. def mainfunction();
3.     global x
4.     print(x)
5.     x = "welcome to Python world"
6.     print(x)
7. mainfunction
8. print(x)
```

~~Literals~~ :-

```
x=100
def main_functions():
    global x
print(x)
x = "Welcome to phython world"
print(x)
main function
print(x)
```

(1) Literal means a value :-

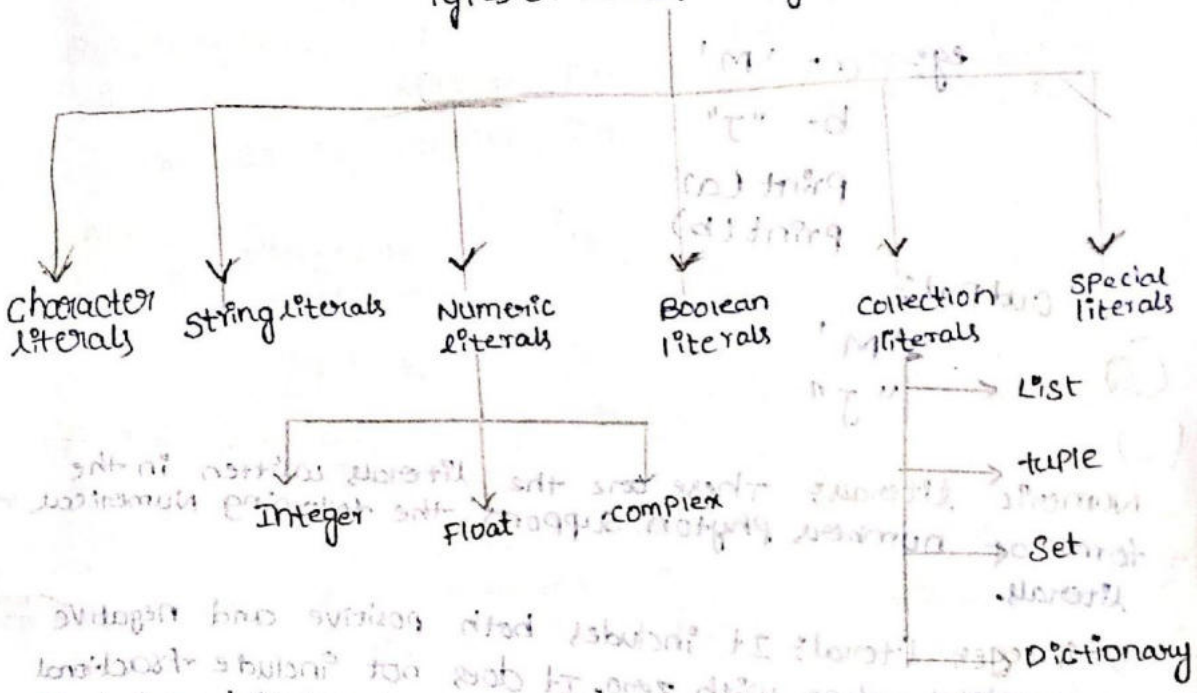


Fig: Literals in phyton

Literals are the fixed values are data items used in a source code. phyton supports different types of literals such as

1. String literals :- The text written in single codes, double, triple codes represents the string literals in phyton. for example "Computer applications", 'V. lakshmi prasanna', etc we can also use triple codes for multly line strings.

program:-

```
a = "Hello"
b = "Prasanna"
c = "Phyton is good for learning platform"
```

```
Print(a)
Print(b)
Print(c)
```

Out Put:-

```
'Hello'
"Prasannd"
''' Python is good for learning Platform'''
```

Character literal:-

It is also a string literal type in which the character is enclosed in single (or) double codes.

```
eg:- a = 'M'
     b = "J"
     print(a)
     print(b)
```

Output:-

```
'M'
"J"
```

Numeric literals:- These are the literals written in the form of numbers. Python supports the following Numerical literals.

1) Integer literal:- It includes both positive and negative numbers along with zero. It does not include fractional parts. It can also have binary, decimal, octal, hexadecimal literals.

2) Float literal:- It includes both positive and negative real numbers.

3) Complex literals:- It includes a+bj a numeral where 'a' represents the real part, 'b' represents the complex part (imaginary part).

```
eg:-
a = 7
b = 8.3
c = -15
Print(a)
Print(b)
Print(c)
```

Output:-

```
a = 7
b = 8.3
c = -15
```

Boolean literals:- The boolean literals have only 2 values in phyton. These are true and false.

```
ex:- a = 3 > 10
     b = (a == 3)
     c = True + 10
     print(a)
     print(b)
     print(c)
```

O/P: False, True, 11

if true it is 1
if false it is 0

Special literals:- phython has a special literal 'None'. It is used to denote nothing, no value, are absence of the value.

```
ex:- var = None
     print(var)
```

O/P:- None

Literal collections:- The 4 literals collection in phyton, are list, tuple, dictionary, set.

List:- It is a list of elements represented in square brackets [ ] with commons in b/w.

* These variables can be of a any data time and can be changed as well

* Tuple:- It is a list of commas seporated elements are values in rounded bracket ( ). The values can be of any data type but cannot be changed.

Dictionary:- It is the unordered set of key value pairs

Set:- It is the unordered collection of elements in curly braces { }.

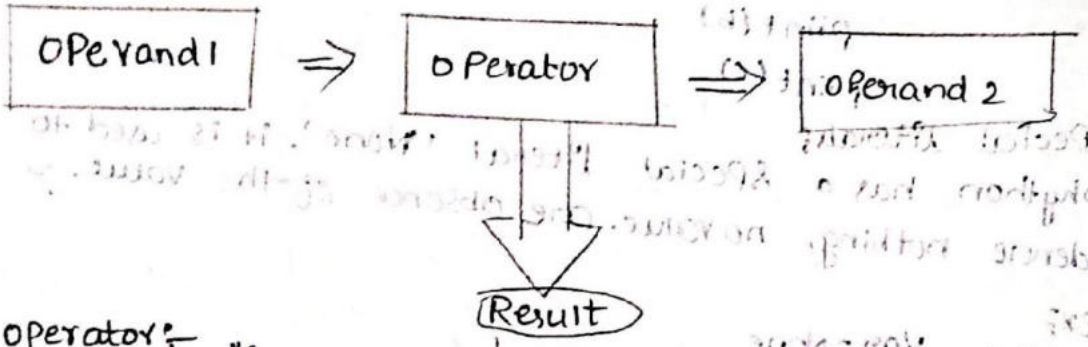
Programs:-

```
My-list = [123, "Joshna", 1.2, 'data']
My-tuple = (1,2,3, 'Janu')
My-dict = {1:'one', 2:'two', 3:'three'}
My-set = {1,2,3,4}
print(My-list)
print(my-tuple)
print(my-dict)
print(my-set)
```

Output:-

```
[123, "Joshna", 1.2, 'data']
(1,2,3, 'Janu')
{1:'one', 2:'two', 3:'three'}
{1,2,3,4}
```

Operators:- Operators in phyton are used for operation b/w two values are variables. The output varies according to the type of operator used in the operation. Operators are the pillars of a program, on which logic is built in a specfic programming language. Suppose if we you whusant to perform two variable values we you can we the addition operator for these operation.



Operator:- ~~Operands~~ It is the special symbols ex:- +, -, *, /, %, ... etc.

Operand:- It is the value on which the operator is applied.

* Depending upon on the type of operations there are 7 types of operators in phyton programming language

1) Arithmetic operator
2) Assignment operator
3) comparsion operator
4) logical operator
5) membership operator
6) Identity operator
7) Bit-wise operator

Arithmetic operators:- they are used to performs arithmetic calculations in phyton.

Name	Symbol
addition	+
subtraction	-
multiplication	*
Division	/
Modulus	%
exponential	**

Program

```
a=10.
b=2
add= a+b
sub= a-b
mul= a*b
div= a/b,
mod= a.1.b.
p = a**b,
```

OIP:-
12
8
20
5
0
100

Assignment Operator:- They are used to calculate or assign value to the variables following are assignment operators that we have in phyton.

Operator	Description
= =	It assign the value of the right expresion to the left operaton
+ =	It increase the value of the left operatand by the value of right operand and assign the modify value back to left operatand.
- =	It decrease the value of the left operand by the value of the right operand and assign the modified value back to the left operand.
* =	It multiples value of the left operand by the value of the right operand and assign the modified value back to the left operand.
/ =	It divides the value of the left operand by the value of right operand
** =	a** =b will be equal to a = a** b for ex:- If a=4, b=2, a** =b assign 4** 2 = 16 to a

Comparision Operators:- they are used to compare two values. following are comparison operators we have in phyton.

Name	Symbol
Equal	==
Not Equal	!=
Greater than	>
less than	<
less than equal to	<=
greater than equal to	>=

ex:-

```
a=10
b=20
Print (a>b)
Print (a<b)
Print (a==b)
Print (a!=b)
print (a>=b)
print (a<=b)
```

IIP

```
= False
True
false
True
false
True
```

logical operator:- They are used primarily in the expression evalution to make a decision. phyton supports the following logical operators.

Name (or) Operation	Symbol (or) Description	Syntax
and	Logical AND: True if both the operands are true	x and y
OR	Logical OR: True if either of the operands is true	x or y
not	logical Not: true if the operand is false	not x

ex:-

```
a = True
b = False
print (a and b)
print (a or b)
print (not a)
```

O/P:-

Membership operator:- It is used to check if a sequence is present in an object. Following are membership operation that we have in python

IN	NOT IN
Returns true if Sequence is present in the object	Return true if sequence not present in the object

example program:-

```
x = 24
y = 20
List = [10,20,30,40,50]
if (x not in list):
    print (" x is NOT present in the given list")
else:
    print (" y is present in the given list")
if (y in list):
    print ("y is present in the given list")
else:
    print (" y is NOT present in the given list")
```

Output:-

x is not present list
y is present in list

Identity operators:-

IS and IS NOT are the identity operators both are used to check if two values are located on the same part of the memory. two variables that are equal do not imply that they are identical.

IS	IS NOT
It is returns true if both variables are same objects	Returns true if both variables are not same objects

example Programs:-

```
a:10
b:20
c:a
print(a is not b)
Print(a is c)
```

O/P:-

True

True

eg: num1:5
num2:5
a=[1,2,3]
b=[1,2,3]
c=a
S_1="helloworld"
S_2="helloworld"
print(num1 is num2)
print(a is b)
print(a is c)
print(S_1 is S_2)

O/P:- True
False
True
True

Bit wise operator:- Bitwise operator perform bit by bit operation on the values of the two operands. following are the bitwise operators that we have in Phyton.

operator	Description
&(bitwise and)	if the both the bits are in two operations 1) then one is occupied to result otherwise zero is copied
| (binary or)	The resulting between is, 0 if both the bits are zero otherwise the resulting bit will be
^(binary XOR)	the resulting bit will be one if both the bits are different otherwise the resulting bit will be zero.
~ (negation)	It calculates the nagition of each bit operator after operator that is if the bit is zero, the resulting bit will be 1
<< left shift	The left operand value is moved left by the ~~number~~ number of bits present in right
>> right shift	The left operand is moved right by the number of bits present in right operand.

Example program:-

```
a=10
b=4
print("a&b=", a&b)
print("a|b=", a|b)
print("a^b=", a^b)
print("~a=", ~b ~a=~b)
print("a<<1=", a<<1)
print("a>>1=", a>>1)
```

O/P:-

```
a&b=0
a|b=14
a^b=14
~a=-11
a<<1=20
a>>1=5
```

Data types:-

Controlled Structures:-

Boolen expressions:- phyton bool() condition is used to return or convert a value to a boolen value that is true (or) false. these values are used to determine the given Statement true(or) false. it denoted by the class bool. true can represented by any non-zero values (or) 'T' where as false can be represented by the 0 (or) 'F'.

Syntax:-

```
bool [[x]]
```

Bool Parameters:- The bool method in general takes on one parameter (herex), which the standard Truth testing proccesor can be applied. we know Parameter is passed by default it returns false. so passing a Parameter is optional. return value bool(): It can rewritten one of the true values.

1) It writtens true, if the parameters (or) value passed is true.
2) It writtens false if the parameters (or) value passed is false. there are a few case in which phyton bools() method returns false. except this all other values return true.
   i) If a false value is passed
   ii) If an empty sequence is passed
   iii) If an empty sequence is passed such as ( ), [ ], etc.

4) If '0' is passed any Numeric 0, 0.0 etc
5) If an empty mapping is passed such as {}
6) If objects of classes having (_bool_ or _len_ methods) returning zero or false.

example Programming:-

```
1 x = False
2 print (bool(x))
3 x = True
4 print( bool (x))
5 x=5
6 y=10
7 print( bool(x==y))
8 x= None
9 print(bool(x))
10 x= ()
11 print(bool(x))
12 x={}
13 print (bool(x))
14 x = 'Hanvitha'
15 print(bool(x))
```

output

```
2 False
4 True
6 False
7 False
9 False
11 False
13 True
```

Selection control Statements:-

In Phyton the selection statements are also known as Decision making statement all are branching Statements. The selection statements are used to select Path of the program to be executed based on the condition. Phyton provides the following selection statements.

1) If statements
2) If else statement
3) If-elif-else statement
4) Nested If Statement.

If Statement:- In Phyton we use if statement to test a condition and decised the executions of a block of statements based on that condition result. The if statements checks the given conditions then decided the execution of a block of statements. If it is true then the block of statements is executed and if false then the block of statements ignored.

Syntax:-

```
if (condition):
    Statement 1
    Statement 2
    Statement 3
```

example program:-

```
age = int (input ("Enter your age:"))
if age <= 18:
print (" you are not eligible for vote").
if age > 18:
print (" you are eligible for vote")
if age >= 60:
print (" you are eligible for senior citizen benifits")
```

OutPut:-

```
enter your age: 65
you are eligible for vote
you are eligible for senior citizen benefits.
```

If else Statement

If we use if else statement to test a condition and pick the execution of a block of statements out of two blocks based on the condition result. The if else statement checks the given condition then decide which block of statements to be executed based on the condition result. If the condition is true then the true block of statements is executed and if it is false then the false block of statements is executed.

Syntax:-

```
if(condition):
    statement(s)
else:
    statement(s)
```

Eg:- num = int (input ("Enter a number of your choice:"))
if (num-1.2==0)
print (" The given number is even number:", num)
else:
print ("The given number is odd number:", num)

O/P:- Enter a number of your choice: 10
even number

3) if-elif statement:- A user can choose if elif statement. A user can choose from a variety of alternatives here. In this varient of if statement there are many else if conditions declared. If the first condition does not satisfy then the next condition is executed. But if this condition also does not satisfy then the next condition is executed.

Syntax:-

```
if condition 1:
    statement (s)
elif condition 2:
    statement (s)
elif condition 3:
    statement (s)
```

else
Statement(s)

ex:
```
marks = int(input("enter marks from 0-50"))
if (marks<20):
print("student failed")
elif (marks>20 and marks<=40)
Print("The student passed with B grade")
else:
Print("The student passed with A grade")
```

O/P: Enter marks from 0-50: 45
the student passed with A grade.

Nested if statement:- A nested if statement is an if statement inside another if statement. It is used when we need to check multiple condition.

Syntax:
```
if (condition 1):
if (condition 2):
else:
Statement 1
else:
Statement 2
```

example program:-
```
num = int(input("Enter a number:"))
if num>0
print("The number is positive")
if num%2==0
Print("The number is even")
if num > 10
print("The number is greater than 10")
else:
print("The number is 10 or less")
else:
Print("The number is odd")
else:
Print("The number is not positive")
```

O/P:
Enter a number: 10
the number is positive
the number is even
the number is 10 or less

Iterative control statements:-
1) while statement
2) for statement

while statement:- The while statement is used to execute a set of statements repeatedly. The while statement is also know as entry control loop statement because in the case of while statement first the given condition is verified then the execution of statement is determined based on the condition result.

Syntax:
```
while condition:
    statement-1
    statement-2
    statement-3
    ---------
    ---------
```

ex:
```
count=0
while (count<9):
Print("The count is:", count).
count = count+1
```
O/P: Print("Good bye user! Job is done").

0 4 8
1 5
2 6 Good bye user! Job is done
3 7

For statement:-
The first statement is used to iterate through a sequence like list, tuple, set, dictionary or string. The first statement is used to repeat the execution of a set of statements for every element of a sequence.

ex program syntax:-
```
for i in range(5)
if i==3
continue
print(i)
```

O/P):
0
1
2
4
5

ex:-
```
for i in range(5)
Print(i)
```
0
1
2
3
4

The iterative statements are also known as looping statement or repetitive statement. The iterative statements are used to execute a path of the program repeatedly as long as the condition is true.

Break and Continue statements :-

1) Break statement

The break statement terminates the loop immediately when it is encounter and break.

ex :-
```
for i in range(5)
    if i == 3:
        break
        print(i)
```
O/P :- 0
1
2

2) Continue statement

The continue statement skip the current iteration of the loop and control flow of the program goes to next iteration. Syntax.

```
continue
```
ex :-
```
for i in range (5)
    if i == 3
        continue
        print (i)
```
O/P :-
0
1
2
4
5

Pass statement :-

The pass statement is a null operation. It does nothing when executed

Ex Prog
```
for i in range(5)
    if i==3
        pass
        Print (i)
```
O/P 1 2 3 4

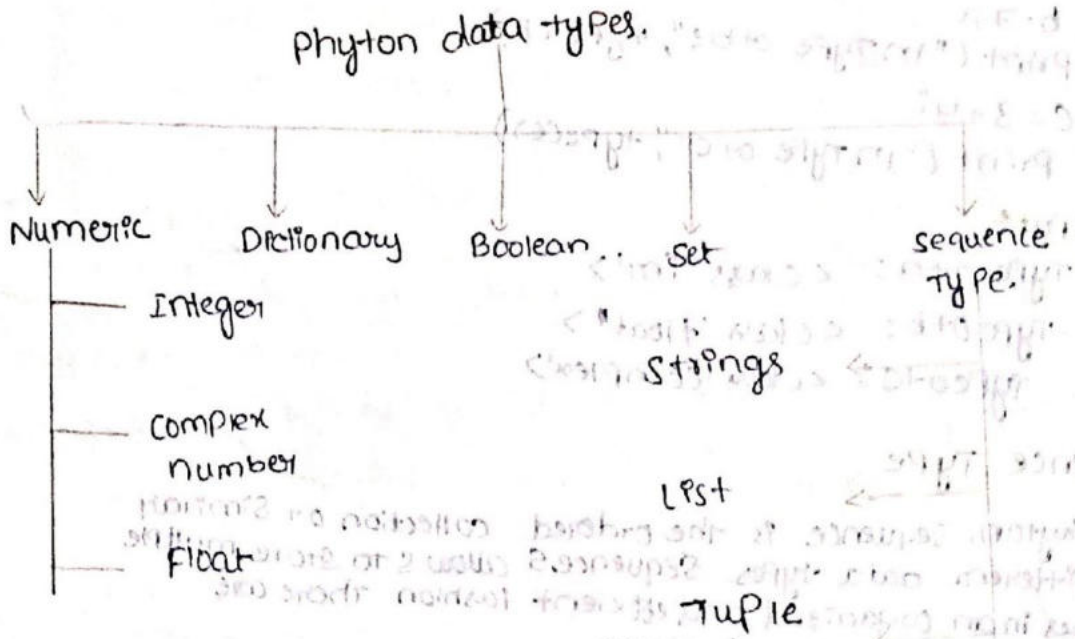
Data types:

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

Variables can hold values, and every value has a data type Python is a dynamically typed language; hence we do not need to define the type of the the variable while declaring it. the interpreter implicitly binds the value with its type.

Python provides various standard data types that define the storage method on each of them. the data types defined in Python are given below.

1. Numbers
2. Sequence Type
3. Boolean
4. Set
5. Dictionary.



Number Stores numeric values. The integer, float and complex values belong to a python Number's data-type. Python provides the type() function to know the data-type of the variable. Similarly, the isinstance() function is used to check an object belongs to a particular class.

Python creates Number objects when a number is assigned to a variable.

Python supports three types of numeric data.

Integers:- This value is represented by int class. It contains + or − whole numbers (without fraction or decimals). Integer value can be any length such as integers is 10, 2, 29, -20, -150 etc. Phyton has no restriction on the length of an integer. Its values belongs to int.

Float:- This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a + or - integer may be appended to specify scientific notation. Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.

Complex numbers:- Complex number is represented by complex class. It is specified as (real part)+(imaginary part)j.

```
#phyton program to
#demonstrate numeric value

a=7
Print("Type of a:", type(a))
b=7.0
Print("In Type of b:", type(b))
c=3+4j
Print("In Type of c:", type(c))
```

Out Put:-

```
Type of a: <class 'int'>
Type of b: <class 'float'>
Type of c: <class 'complex'>
```

Sequence Type

In phyton, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in phyton.

* String
* List
* Tuple

String:-

In Phyton, strings are arrays of bytes representing unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In Phyton there is no character data type, a character is a string of length one. It is represented by str class.

In the case of string handling, the operator + is used to concatenate two strings as the operation "hello" + "Phyton" returns "hello Phyton".

The operator is known as a repetition operator as the operation "phyton"* returns "pythonpython".

Creating string: Strings in Phyton, can be created using single quotes or double quotes or even triple quotes.

ex:-

```
# with Single Quotes
String1 = 'Welcome to AITS'
print("String with use of Single Quotes:")
print(String1)
# creating a string
# with double Quotes
String1 = "I'm a milky"
print("In string with the use of Double Quotes:")
print(String1)
print(type(String1))
# Creating a string
# With triple Quotes
String1 = '''I'm a milky and I live in a world of
"phyton"'''
print("In String with the use of Triple Quotes:")
print(String1)
print(type(String1))
```

Out Put:-

```
String with the use of Single Quotes:
Welcome to AITS
String with the use of Double Quotes:
I'm a milky
<class 'str'>
String with the use of triple Quotes:
I'm a milky and I live in a world of "phyton"
<class 'str'>
```

List:- Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are seperated with a comma (,) and enclosed within square brackets [ ]

We can use slice [:] operations to access the data of the list. The concatenation operator (+) and repetition operator (*) works with the list in the

same way as they are were working with the list.

ex:-
```
# Python program to demonstrate creation of
list

List = []
print ("Initial blank List:")
print (List)
List = ['MILKY']
print ("In List with the use of String:")
print (List)
list = ["A", "for", "Apple"]
print ("In List containing multiple
                values:")
print (list[0])
print (list[2])
Initial blank List:
[]
List with the use of String:
['MILKY']

List containing multiple values:
A
for
Apple >>>
```

Tuple: A tuple is similar to the list in many ways like lists, tuples also contains a collection of the items of different data types. the items of the tuple are seperated with a comma (,) and enclosed in parentheses (), it is represented by tuple class. A tuple is a read only data structure as we can't modify the size and value of the items of a tuple.

:-
```
# Python program to demonstrate creation of set
# Creating an empty tuple.
tuple1 = ()
print ("Initial empty Tuple:")
print (Tuple1)
# Creating a Tuple with the use of String
Tuple1 = ('Milky', 'honey')
print ("In Tuple with the use of String:")
print (Tuple1)
# Creating a Tuple with the use of list
list1 = [1,2,4,5,6]
print ("In Tuple using List:")
print (tuple(list1))
Initial empty Tuple:
()
Tuple with the use of String:
('Milky', 'honey')
Tuple using List:
(1,2,4,5,6)
>>>
```

Boolean: Boolean type provides two built-in values, True and false. these values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 (or) 'F'. It is denoted by the class bool.

Note: True and false with capital 'T' or 'f' are valid booleans otherwise Python will throw an error.

```
# Python program to demonstrate boolean type
print (type(True))
print (type(False))
print (type(true))
<class 'bool'>
<class 'bool'>
Traceback (most recent call last):
Print (type (true)),
```

Name error:- name 'true' is not defined

```
>>>
```

Set: Phyton set is the unordered collection of the data type. it is iterable, mutable, and has unique elements in set, the order of the elements is undefine d; it may return the changed sequence of the element. the set is created by using a built in function set (), or a sequence of elements is passed in the curly braces and separted by the comma. it can contain various types of values.

ex:-
```
# Creating empty set
Set1 = set()
Set2 = {'Hanvitha', 2, 3, 'Python'}
# Printing Set value
print (Set2)
# Adding elemement to the Set
Set2.add(10)
print (Set2)
#Removing element from the set
Set2.remove(2)
Print (Set2)
```

Output : {'Hanvitha', 2.3, 'Python'}
{2.3, 'Hanvitha', 10, 'Python'}
{3, 'Hanvitha', 10, 'Python'}
>>>

Dictionary: Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

Creating Dictionary: In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by 'comma'. Values in a dictionary can be of any data type and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built in function dict(). An empty dictionary can be created by just placing it to curly braces {}

ex:- d = {1: 'milky', 2: 'Hanvitha', 3: 'krishna', 4: 'Sweety'}
# Printing dictionary
print(d)
# Accessing value using keys
print("1st name is", d[1])
print("2nd name is:", d[4])
print(d.keys())
print(d.values())

Out Put: {1: 'milky', 2: 'Hanvitha', 3: 'krishna', 4: 'Sweety'}
1st name is milky
2nd name is sweety
dict_keys([1, 2, 3, 4])
dict_values(['milky', 'Hanvitha', 'krishna', 'sweety'])
>>>

Python Variables: Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a inferred language and smart enough to get variable type. Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for the Variable name. milky and milky both are two different Variables.

Rules for creating variables in Python:

Variables are the example of identifiers. An identifier is used to identify the literals used in the program.

The rules to name an identifier are given below

* The first character of the variable must be an alphabet or underscore (_)
* All the characters except the first character may be an alphabet of lowercase (a-z), uppercase (A-Z), underscore, or digit (0-9)
* Identifier name must not contain any white-space or special character (!, @, #, %, ^, &, *)
* Identifier name must not be any word defined in th

Unit-

List:- A list is in phyton is used to store the sequence of various types of data. Phyton lists are mutable type it means we can modify its elements after it created. however Phyton consists six data types that are cable to store the sequences but the most common and reliable type is the list.

* A list can be defined as a collection of values or items of different types. the items in the list are separated with comma, and enclosed with square brackets []. The list can be classified into the following or structure of phyton list.

Syntax:- ~~By using square bracket~~

List-name = [element-1, element-2, element3]

ex:-

emp_data = [101, 'krishna', 20000]

print (emp_data)

By using constructor

O/P:- 101, 'krishna', 2000.0

* The list has the following characters
1) The list are ordered
2) The elements of the list can be accessed by index.
3) the list are the mutable type
4) A list can store the number of various elements.

Creating a list in Phyton:-

list in phyton can be created just placing the sequence inside the square bracket [].

* the general syntax for creating a list is as follows.

Syntax:-

List-name = [element-1, element-2, element-3]

* In phyton. a list can also be created using constructor which stores the details of student or employee.

(or It takes only one argument.

for example consider the following code for creating a list using list() constructor, which stores the details of the student.

ex:- By using constructor

Student_info = list([1, 'Thanvika', 'B-Tech', 89.0])

print(type(student_info))

print(student_info)

O/P:-

<class 'list'>

[1, 'thanvika', 'B-Tech', 89.0]

creating a list with repeated elements by using multiplication method.

we can create a list with reapted elements using the multiplication operator.

ex:- a = [2]*5

ex:- b = [0]*7

print(a)

print(b)

O/P:- [2,2,2,2,2]

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

By accessing elements of a list.

elements in a list can be accessed using indexing. Phyton indexs starts at 0, so a[0] will acess the first element, while negative indexing allows us to acess elements from the end of the list. like index -1 represents the last elements of list.

Syntax:-

List_name [index]

ex:-

a = [10, 20, 30, 40, 50, 60]

print(a[0])

print(a[-1])

print(a[2])

O/P:- 10

60

30

List indexing and slicing:-

to find the index of an element in a list, and explore different scenario will using list index method

next index method search for a given a element from a start of the list and return the position of the first occurence.

Syntax:-

```
list-name.index(element, start, end)
```

example program:-

```
a=["cat","dog","Monkey","Donkey","Tiger"]
print(a.index("Monkey"))
print(a.index("Tiger"))
print(a.index("cat"))
```

o/p:- 2
4
0

Slicing (or) Spliting:- List Slicing allows you to exctract a portion of a list by specticing the start index, stop index, step

Syntax:-

```
List[start:stop:step]
```

ex:-

* The start denotes the starting index position of a list
* The stop denotes the last index position of a list
* The step is used to skip the nth element with in a start:stop

1:3

ex:-

```
num=[0,1,2,3,4,5,6,7,8,9]
print(num[2:5])
print(num[:4])
print(num[5:])
print(num[::2])
print(num[::-1])
print(num[2::])
```

o/p

2,3,4

0,1,2,3

5,6,7,8,9

0,2,4,6,8

9,8,7,6,5,4,3,2,1,0

2,3,4,5,6,7,8

Adding elements to the list:-

we can add elements into the list by using the following methods

1) append()
2) Insert()
3) extend()

1) append() by using append method we can add only one element at the end of list

Syntax:- Listname.append()

ex:-

```
List=[1,2,3,4]
List.append(5)
print(list)
```

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

2) Insert():- by using Insert method we can add element at specified position.

Syntax:- List-name.insert(position, element)

ex:-

```
List=[1,2,3,4,5]
list.insert[3,9]
print(list)
```

o/p:- [1,2,3,9,4,5]

3) Extend:- By using extend method we can add multiple elements at the end of the list.

Syntax:- list-name.extend(elements)

ex:-

```
List=[1,2,4,5]
list.extend((4,5,))
print(list)
```

o/p:- (1,2,4,5,4,5)

Remove elements of the list from:-

We can remove elements from the list by using the following methods

1) remove()
2) pop()
3) delete()
4) clear()

1) Remove:- By using remove method we can remove element based on the first occurence

Syntax:-

list-name.remove(element)

ex:-

```
List = [1,2,3]
list.remove(2)
print(list)
```

O/P:- [1,3]

ex.2

```
List = [1,2,3,2,4,2,5,2]
list.remove(2)
print(list)
```

O/P:- [1,3,2,4,2,5,2]

2) POP:- by using this method we can delete the element based on the specified index. If no index is specified then the end of element will be deleted.

Syntax:-

List-name.POP(index)

ex:-

```
List = [1,2,3,4]
list.POP(2)
print(list)
```

O/P:- [1,2,4]

ex.2:-

```
list = [1,2,3,4]
list.POP()
print(list)
```

O/P:- [1,2,3]

3) Delete:- By using delete method we can delete element based on the index number

Syntax:-

Delete list-name(index)

ex:-

```
List [1,2,3,4]
del list(1)
print(list)
```

O/P:- [1,3,4]

4) clear() By using this method we can clear all the element which are present in the list.

Syntax:

```
list-name.clear()
```

```
list = [1,2,3,4,5,6]
list.clear()
print(list)
```

O/P:- []

More on Phython list

1) replace(). By using replace method we can replace the elements in the list where

Syntax:- List.name.replace(old, new, count)

```
List = [1,2,3,4,5]
list[3] = 8
print(list)
```

O/P:- [1,2,3,8,5]

ex:- 2

```
text = "Hello world"
newtext = text.replace("world", "python")
print(newtext)
```

O/P:- Hello python

How

2) count() By using count method we can count the specific element present in a list.

Syntax:-

```
list-name.count(element)
```

ex:-

```
list = [1,2,4,5,2,3,2]
print(list.count(2))
```

O/P:- 3

3) len() The len() method returns the number of items present in the list

Syntax:-

```
len(iterable)
```

```
numbers = [1,2,3,4,5,6,7,8,9]
print(len(list))
```

O/P:- 9

4) max() This method is used to find out the maximum element which are present in the list.

Syntax:

```
max(iterable)
```

ex:-

```
number = [10,20,30,40,50]
Print (max(number))
```

O/P:- 50

5) min ( ) :- this method is used to find out the minimum element which are present in the list.

min (iterable) :- syntax

ex:- number = [10, 20, 30, 40, 50]
Print (min (number))

O/P:- 10

6) sum ( ) :- this method is used to find out the total sum of element which are present in the list.

Syntax:-

```
min(
Sum (iterable, start=0)
```

ex:- number = [10, 20, 30, 40, 50]
print (sum(number)
O/P:- 150

7) Sort ( ) :- By using this method the elements sorted either ascending (or) descending

Syntax:-

List-name.sort (reverse = False)

ex:-

```
number = [4,2,9,1,5]
Numbers.sort ()
Print (numbers)
O/P:- [1,2,4,5,9]
```

ex:-

```
numbers = [4,2,9,1,5]
```

8) Sorting in Descending order:-

```
numbers=[4,2,9,1,5]
Numbers.sort (reverse = True)
print(numbers)
```

O/P:- [9,5,4,2,1]

9) Assigning values:- We can assign a list to use an another list

Syntax:-

variable1, variable2 = value1, value2

ex:- X, Y, Z = [10,20,30]
Print (X, Y, Z)

O/P:- 10,20,30

ex:-2:-

```
List = [10,20,30,40]
new = list
print(list)
print(new)
```

O/P:- [10,20,30,40]
[10,20,30,40]

10)10) copy() :- It is used to copy the entire list to another list. here the list cannot be changed even if we have changes another.

syntax:-

```
new_list = old_list.copy()
```

ex:-

```
original_list = [1,2,3,4]
copied_list = original_list.copy()
print(copied_list)
```

O/P:- [1,2,3,4]

ex:-2

```
L= [10,20,30,40]
new = L.copy()
print(L)
print(new)
L(2) = 50
print(L)
print(new)
```

O/P:- [10,20,30,40]
[10,20,30,40]

11) all() :- This method is used to check if all the elements are true other wise then it will retrun true.

Syntax:-

```
all(iterable)
```

ex:- L= [1,2,0,3]
print all(L)

O/P:- False

ex:-2

```
List = ["true",1, "hello"]
print(all(list)
```

O/P:- True.

2) any() :- It returns true if atleast one element is true which are present in the list otherwise it return false

Syntax:-

any(iterable)

ex:-1

```
List = [1,0]
print(any(list))
```

o/p:- True

ex:-2

```
List = [0, False]
print(any(List))
```

o/p:- False.

3) Enumerate:- This method is used to track the index

Syntax:-

enumerate(iterable, start=0)

The enumerate function is used to track of the index the values which are present in the list.

ex:-

```
enumerate(iterable, start=0)
branch = ["CSE", "AI&DS", "ECE"]
for i, branch in enumerate(branch)
print[f"{i}: {branch}"]
```

o/p:- 0: CSE
1: AI &DS
2: ECE

ex:-2

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
print(index, fruit)
```

0 apple
1 banana
2 cherry

chr() ⇒ character.

It is used to convert the Ascii code into characteristics

Syntax:-

chr(numbers).

A = 65
A = 96

A-65
a-97

ex:-

```
ascii_value = 65
chr = chr(ascii_value)
print(chr)
```

O/P:- 'A'

ex:- 2.

```
ascii_list = [65, 66, 67, 68]
chr_list = [chr(num) for num in ascii_list]
Print(chor_list)
```

O/P:-

['A', 'B', 'C', 'D']

order:- Ord() ⇒ order:- It is used to convert characteristics into their Ascii values by using the order

Syntax:-

~~order~~

Ord(characters)

ex:-1

```
char = 'A'
ascii_value = ord(char)
print(ascii_value)
```

O/P:- 65

ex:2) Convert list of characters into Ascii_values :-

```
Char_list = ['a', 'b', 'c', 'd']
ascii_list = [ord(char) for char in Char_list]
print(ascii_list)
```

O/P:- 97, 98, 99, 100

ex:-3 Converting a string into Ascii_list

ex:-

```
text = "Hello"
ascii_values = [ord(char) for char in text]
print(asii_values)
```

O/P:- [72, 101, 108, 108, 111]

Iterating over list in python:- Iterating over list in phyton provides a several ways to iterate over list. the simpliset and common way to iterate over list by using for loop.

For loop :- This method allows us to access each element in the list directly

ex: Program

```
a=[1,3,5,7,9]
for i in a:
    print(i)
```

i :- index.

O/P :- 1
3
5
7
9

Ex:-2 Sum of elements using for loop

```
num = [1,2,3,4,5]
sum = 0
for i in num:
    sum += i
    print("sum is:", sum)
```

O/P :- The sum is: 1
The sum is: 3
The sum is: 6
The sum is: 10
The sum is: 15

while LOOP( ) :- This method is similar to for loop method. We need to find out the length of the list by using len( ), start (or) index at '0' and access each item by its index the increment by: 1

```
L=[10,20,30]
i=0
while i<(len(L)):
    print(L[i])
    i+=1
```

O/P :- 10
20
30

Enumerate :-

This method provides both index(i) and value of each element

ex:- a=[1,3,5,7,9]

```
for i, value in enumerate(a):
    print(i, value)
```

O/P :- 0 1
1 3
2 5
3 7
4 9

Set :- A set is a built in data type that represent unorder collection of unique elements.

* It is useful when you need to store distinct values and perform mathematical set operations like union, intersect and difference.
* A set can be created with the help of unorder elements which are separated with commas and enclosed with curly braces ({ }) (or) constructor.

Key characteristics of a phyton set :-

1) Unordered :- The elements have no fixed order
2) Unique elements :- A set automatically removes duplicates.
3) Mutable :- You can add (or) remove elements.
4) Unindexed :- you cannot access elements using an index.
5) creating a set :- you can create a set using curly braces (or) and the set constructor

ex:- My_set = {1,2,3,4,5}
print(my_set)
O/P :- {1,2,3,4,5}

Syntax

set_name = {elements,}

creating a set with duplicate values :-

ex:- duplicate_set = {1,2,2,3,4,5,5}
print(duplicate_set)
O/P :- {1,2,3,4,5}

we can remove duplicate values by using duplicate method.

creating an empty :- we can create empty set by using this method

ex:- empty_set = ( )
print(empty_set)
O/P :- set()

Adding and removing elements in a set :- we can modify sets by using the following methods

```
# Define a set
numbers = {1,2,3,4,5}
# Add elements
```

we can add elements into the set(by using add method

```
numbers = {1,2,3,4,5}
add numbers(5)
print(numbers)
```

O/P :- {1,2,3,4,5}

remove() elements :-

We can remove elements from the set by using remove method.

```
numbers = {1,2,3,4,5}
remove.numbers(2)
print(numbers)
```

O/P :- {1,3,4,5}

Discard() :- We can discard the elements by using discard.

```
numbers = {1,2,3,4,5}
remove.numbers(6)
print(numbers)
```

O/P :- error

ex :-
```
numbers = {1,2,3,4,5}
discard numbers(6)
print(numbers)
```

O/P :- {1,2,3,4,5}

ex :-
```
numbers = {1,2,3,4,5}
discard numbers(2)
print(numbers)
```

O/P :- 1,3,4,5

This method is different from the remove method because the remove method will raise an error if the specified item does not exist and discard method will not.

Syntax :-

set.discard(values)

Clear() :- By using this methods we can clear() or remove all the elements in the set.

ex :-
```
numbers = {1,2,3,4}
clear.numbers()
print(numbers)
```

O/P :- set()

pop() :- By using this method we can remove specific element.

ex :-
```
numbers = {1,2,3,4,5}
pop.numbers(0)
print(numbers)
```

O/P :- {1,2,3,4}

Mathematical set operations

1) union
2) Symmetry
3) Intersection
4) subset
5) Difference

Python provides several built-in set operations that all us to perform mathematical set operation like union, intersection, difference and soon.

1) union : (| or union())

The union combines elements from both the sets

ex :-
```
A = {1,2,3,4}
B = {3,4,5,6}
print(A|B)
print(A.union(B))
```

O/P :- {1,2,3,4,5,6}

2) Intersection : (& or intersection())

It is used to find the common elements

```
A = {1,2,3,4}
B = {3,4,5,6}
print(A&B)
print(A.intersection(B))
```

O/P :- {3,4}

3) Difference :- (- or difference())

It is used to find out the elements which are presents in set A but not find in set B

```
A = {1,2,3,4}
B = {3,4,5,6}
print(A-B)
print(A.difference(B))
```

O/P :- {1,2}

4) Symmetric difference :- (^ or symmetric difference())

elements in a or b which are present in both the sets.

ex: A = {1,2,3,4}
B = {3,4,5,6}
print(A^B)
print(A.symmetric difference(B))

O/P: {1,2,5,6}

Subset: (<= or issubset())

A is a subset of B if all elements of A are in B

ex: A = {1,2,3,4}
B = {3,4,5,6}
C = {1,2,3}
print(C<=A) (or) print(C.issubset(A))

O/P: True.

Superset: (>= or .issuperset())

check if A contains all the elements of C

ex: A = {1,2,3,4,5}
B = {3,4,5,6}
C = {1,2,3}
print(A>=C)
print(A.issuperset(C))

O/P: True
True

Disjoint set:

~~Disjoint set(~~

check if A and B no elements in common

~~Disjoint set~~

A = {1,2,3,4}
B = {5,6,7,8}
print(A.isdisjoint(B))

O/P: True.

What is Tuple? Discuss about tuple.

Tuple is a immutable data-type it means we cannot change after tuple is created tuple is a collection of objects enclosed within parantheses and seperated by comma.

* A Tuple is similar to Python list in terms of indexing and nested objects and also repetition
* The main difference is python tuple is immutable and the Python list is mutable.
* Tuples are ordered and we can access their elements using index values.
* We cannot update items to a tuple once it is created.
* Tuples can't be appended or extended.
* We cannot remove items from a tuple once it is created

Characteristics:

Immutable: Tuple elements can't be modified

Ordered: Maintains specific order for their elements

Indexed: elements are accesed by using respective index.

Creating a tuple:

```
t=()
Print(t)
t=(1,2,3)
print(t)
```

we can use the type() method to know the type of elements

```
t=(1,2,3)
print(type(t))
```

Tuple of Integer

```
t=(1,2,3,4)
print(t)
```

Tuple of string

```
t=('Ese', "BEEE')
print(t)
```

Tuples of mixed datatypes

```
t=(9,'Blue',102)
Prin(t)
```

Tuple of nested tuple.

```
t=((1,2),(me,you))
Prin(t)
```

Tuple operations:

Indexing:-

Phython index start with zero will access the first element of tuple and negative index start with -1 will access the last element of the tuple.

```
t= (1,2,3,4,5)
print (t[1]) 2
Print (t[-2]) 4
print (t[0]) 1
pint [t[-1]) 5
```

Slicing Used to access elements from index to another index

Syntax:- tuple [start: stop: step]

start - starting element of the tuple.
stop - last element
step - how many elements to be ~~stop~~SKIP.

```
t= (0,2,4,'hi','kate','Bye',9.9,'cse',3.9)
print (t[1:4])
Print (t[3:8])
print (t[-4:-1])
print (t[::3])
print(t[::-1])
print(t[::2])
print (t[-6:-3])
```

2,4,hi
hi, kate, Bye, 9.9, CSE
Bye, 9.9, CSE, 3.9
hi, 4, 2, 0
9, 3, CSE, 9.9, Bye, kate, hi
hi, kate, Bye

Concatination

By using '+' operator

```
t1= (1,3,5)
t2= (2,4,6)
print(t1+t2)
```

1,3,5,2,4,6

Repetition

- By using '*' operator

```
t=('Iris')*3
print (t)
```

Iris
Iris
Iris

Deleting The tuple elements can't be deleted by 'del' keyword To delete an entire tuple, we can use delete keyword with tuple name.

```
t= (1,2,3)
del t
t=(1,2,3,4)
```

If you want to delete specific elements convert tuple to list and revers after modification.

```
l = list(t)
print(l)  # 1,2,3,4
l.remove(3)
print(l) # 1,2,4
t = tuple(l)
print(t)  # 1,2,4
```

TUPle methods

1. count
   → To count no. of specific elements

syntax:- tuple name.count(element)

ex:-
```
t= (2,5,8,5,4,2,5,8)
print(t.count(5))
```
O/P :- 3

2. Index
   to know the index of element

tuple name.index(element)

ex:-
```
t=(8,7,6,5)
print(t.index(6)) # 2
```

TUple functions

1. len() No. of elements in a tuple

ex:-
```
t=(9,8,7,6,5)
print(len(t)) # 5
```
output

2. max() To find the maximum element in a tuple.

ex:-
```
t=(8,7,9,5)
print(max(t))
```
output :- 9

3. min() TO find minimum element in a tuple

ex:-
```
t= (8,7,9,5)
print(min(t)
```
Output :- 5

4. Sum()
returns sum of the elements
ex:- t = (9,8,9,2,4,3,6)
print (sum(t))
Output: 41

5. all()
Returns true if all elements are true
t = (1,0,true)
print (all (t)) # false
t = (1, True, 'CSE')
print (all(t)) # True

6. any() returns true if any one element is true
ex:- t = (False, 0, 1)
print (any (t))
output : True
t = (Name, False, 0)
print (any (t))
# False
t = (Name, False, 1)
print (any (t))
True

Dictionary :- A dictionary is a muttable. It is unordered
Collection of key value pair of elements. we can access the values
or elements based on that key only. we cannot use index
value. Keys are unique. there are no duplicate keys.

characteristics of dictionary :-

1) Key value Pairs: Each item in a dictionary is stored has key value pair.
2) Unordered { key : value }
3) Unique elements: Duplicate keys are not allowed.
4) Muttable: you can add (or) update (or) Remove key value pairs.
5) Any data type: Both keys and values can be different data types.
6) Support nesting: Dictionaries can contain other dictionary list.

Creating a dictionary
Dictionary can be created by placing sequence of elements within curly braces and separted with comma

Program:-
d = { }
print (type(d))
O/P:- <class 'dict'>

In dictionary keys and values are separted by colon symbol
Ex:- dic = {'college': 'AU', 'Branch': 'AI&DS', 'section': 'A'}
print (dict)
O/P:- {'college': 'AU', 'Branch': 'AI&DS', 'section': 'A'}

Accessing we can acess the dictionary values based on their keys
ex:- dic = {'college': 'AU', 'Branch': 'AI&DS', 'section': 'A'}
print (dict ['college'])
O/P:- AU

get () :- same as accesing

ex:-

```
dict = {"college": 'AU', 'Branch': 'AI&DS', 'Section': 'A'}
print(dict.get['college'])
```

O/P:- AU

We can also use get method to acess dictionarie values based on keys

Dictionary methods:-

1) Adding elements into dictionary:- We can added an element into the dictionary by using add method

ex:-

```
dict = {1: 'Red', 2: 'Black', 3: 'Blue'}
dict[4] = 'Green'
print(dict)
```

O/P:- {1: 'Red', 2: 'Black', 3: 'Blue', 4: 'Green'}

2) Update elements | key pair into the dict:-

ex:-

```
dict = {1: 'Red', 2: 'Black', 3: 'Blue'}
dict.update({2: 'white'})
print(dict)
```

O/P:- {1: 'Red', 2: 'white', 3: 'Blue'}

3) Remove elements from the ~~list~~ dict:-

We can remove elements by using different methods

1) delete () method:- by using delete method we can delete particulary element based on key.

```
dict = {1: 'Red', 2: 'Black', 3: 'Blue'}
del dict[3]
print(dict)
```

O/P:- {1: 'Red', 2: 'Black'}

2) Pop () method:- By using 'pop' method we can delete from dictionary element

```
dict = {1: 'Red', 2: 'Black', 3: 'Blue', 4: 'Pink'}
dict.pop(3)
print(dict)
```

O/P:- {1: 'Red', 2: 'Black', 4: 'Pink'}

Pop Item:-

```
dict = {1: 'Red', 2: 'Black', 3: 'Blue', 4: 'pink'}
dict.popitem()
print(dict)
```

O/P:- {1: 'Red', 2: 'Black', 3: 'Blue'}

Clear ():- dict = {1: 'Red', 2: 'Black'}

Clear method is used all the elements which are present in the dictionary

```
dic = {1: 'Red', 2: 'Black', 3: 'Blue', 4: 'Pink'}
dic.clear()
print(dict)
```

O/P:- {}

Functions:-

Functions are classified into the following

d = {1: 'Red'}

1) keys:- By using this fun-x we can display all the key from the dictionarie.

ex:- d = {1: 'Red', 2: 'Black', 3: 'Green', 4: 'pink'}
print(d.keys())

O/P:- dict_keys([1,2,3,4])

2) values:- By using this fun-x we can display the all the values from the dictionary.

O/P ex:- dict_values(['Red', 'Black', 'Green', 'Pink'])
print(d.values())

3) Items: By using this fun-x we can displays all the items which are present in the dictionary.

O/P ex:- dict_items([(1: 'Red'), (2: 'Black'), (3: 'Green'), (4: 'Pink')])
print(d.items())

4) Len:- By using this fun-x we can find the length of the dictionary

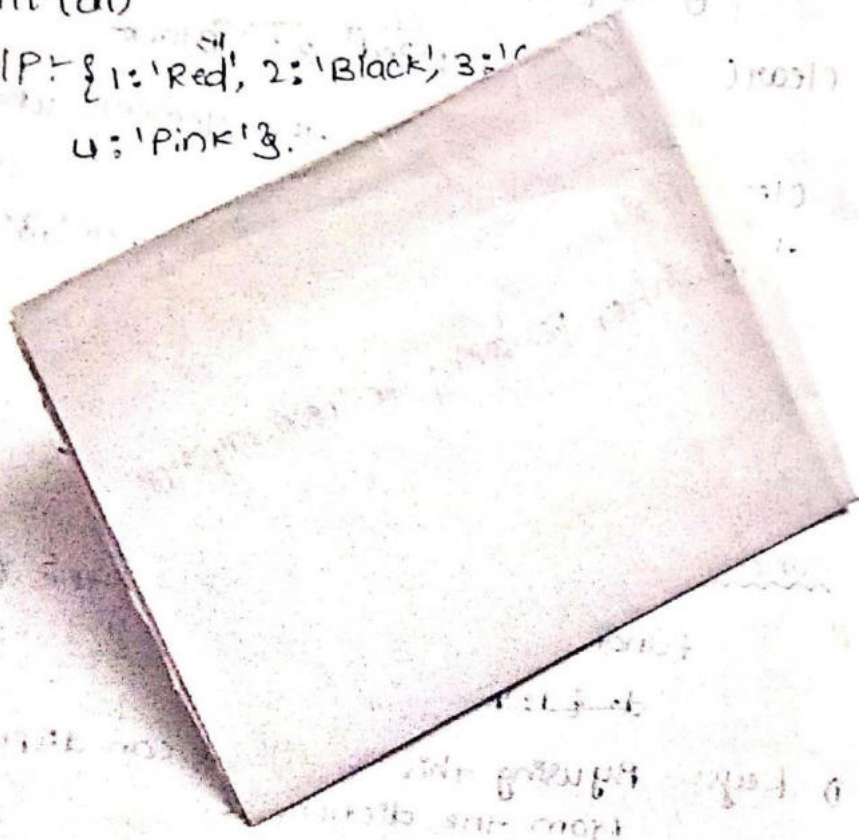
ex:- d = {1: 'Red', 2: 'Black', 3: 'Green', 4: 'Pink'}
print(len(d)) O/P:- 4

copy:- By using this fun^x we can copy the data (or) elements from one dictionary to another dictionary

ex:-

```
d= {1:'Red', 2:'Black', 3:'Green', 4:'Pink'}
d1 = d.copy()
print(d1)
```

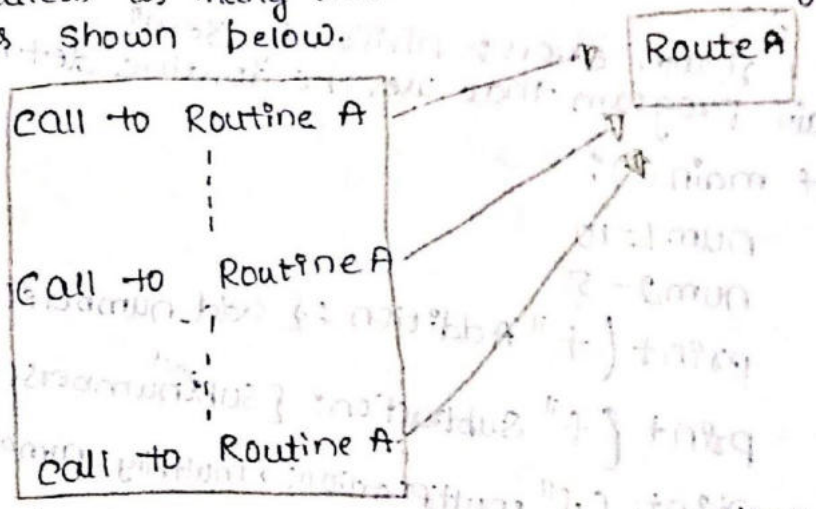
O/P:- {1:'Red', 2:'Black', 3:'
4:'Pink'}.



Unit-3

Functions

Program routines:- A routine is a named group of instructions performing some task. A routine can be invoked (called) as many times as needed in a given provoke as shown below.



when a routine terminate execution automatically returns to call from which it was called. such routines may be predifend in the programming language or designed and implemented by the programm -er

* In phyton a routine refers to a function or method that performs a specic task when called. routines helps in bracking down a program into smaller, manggble tasks to improve code reusablity, readbility and mantanability

Here is an example of a simple program with routines in Phyton:

```
# Routine 1: A function to add two numbers
def add_numbers(a,b):
    Return a+b

# Routine 2: A function to subtract two numbers
def subtract_numbers(a,b):
    Return a-b

# Routine 3: A function to multiply two numbers
def multiply_numbers(a,b):
Return a*b
```

Routine 4: A function to divide two numbers

```
def divide_numbers(a,b):
    return it
    if b!=0
    return a/b
    else
    return "errors: Division by zero"
```

main program that uses the routine def main():

```
def main():
    num1=10
    num2= 5
    print(f"Addition :{ add_numbers(num1,num2)}")
    print(f" Subtraction: { subtract_numbers(num1, num2)}")
    print(f" multiplication: { multpliy_numbers(num1,num2)}")
    print(f"division: { div_numbers(num1,num2)}")
if__name__=="__main__":
    main()
```

opp:- Addition: 15
subtraction: 5
multiplication: 50
division: 2.0

if__name__=="__main__": It is the statement or constructur in phyton checks it current script is being done directly as the main program, (or) it it is being imported module into another program.

__name__ is a variable that exists in every phyton module and is set to name of the module.

Basic Syntax of f-string it was invented in phyton3.6 version

To use an f string, prefix a string with f and enclosed with curly brackets expression inside curly brackets.

f-strings.(formatted string literals) were introduced in phyton 3.6 and proved a readable and effecient way to format strings.

Advantages of f-strings

1) Readability
2) support expressions inside { }
3) support function calls inside { }
4) support formatting numbers
5) support padding and alignment.

functions :- Python functions are simple to defined and essential to intermediate level programming the extact hold to function as they do too variables. The goal is to group up certain often performed actions and define a function.

rather than rewriting the same code block over and over varied input variables, we may call the function and re purpose the code included within different variables.

- A function can be related to the purpose of making tea. If you were to make tea for the first time your parents would have define the steps to make. the next time they want to drink they will call you directly by saying make tea. similarly in the case of fun-x you first define the sequence of steps that you would like to carry out to achieve a certain task. later you can call out fun-x by its name and these steps will be formed.

Definition :- A fun-x is a block of code that Performs a specific task it allows us to reuse code, making programs more organised, modular and efficient

- in other words a fun-x is a block of code that performs a specific and well defined task. it organises the code into a logical way to perform a certain task.
- fun-x is syntax of structure in which we represent the reusable buisness logic.
- every fun-x is called by a certain name and block of code inside the fun-x gets executed when you call the fun-x

It we have a block of code that gets invoked more than one's put into a fun-x this is because fun-x allows us to use a Block of code repeatedly in different sections of a Program.

- It makes the program more efficient by minimising the repetition. fun-x helps to break down long and complex problems into smaller and managable segment and enhance program readability
- They provide better modularity for your application Program and high reusability code. sometimes we can the fun-x with another name call method. But there is a slight difference repeatdly biwa fun-x. method.
- fun-x are defined outside the class where as methods are defined inside the class

Syntax:-

Syntax to defined user defined fun-x in phyton. + by user creates or defindes user defined fun-x the general Syntax to declare a user defined in fun-x in Phyton is as follows.

Syntax:-

```
def function_name (Parameters):
    """ doc string """
    ------
    ------
    ------
    return [expression or values]
```

In the above Syntax the first line of fun-x definition is header, and the rest is the body of the fun-x whereas the fun-x definition begins with "def" key word followed by function name, parathasis() and then (:). The parameter list placed within the parantheis is optional.

Advantages of fun-x in python:-

- In fun-x we can prevent repting the same code repeatdy in a program
- python fun-x one's defined can be called many times and from any where in a program.

3. In our python program is large it can be sparted into numerous functions which is simply to track the
4. the key accomplishment of a python fun-x is we can create as many outputs as we want with different arguments.

keyword def:

function defination starts with the keyword, def, that defines the function it tells the beginning of the function headers

Function name:

The function name represent the name of a function.

Parameter:

The parameters (arguments) placed inside to paranthesis on optional

* A parameter is a information that we use inside the fun-x to perform a spetic task
* we can represent more parameter separted by comma inside the paranthesis

Colon:-

A colon indicates end of fun-x headers

Doc string:

A docstring is a documentation string that is an optional components we commonly use in to describe what fun-x does,

→ It is enclosed by (""" """)

Statement:

The body of fun-x consist of one or more valid statement - each statement must be intected same indentation to form a block.

return statement:

A return statement ends to Fun-x call and returns a value from a function back to the fun-x calling code It is optional

4. Recursive function

It is a fun^n that calls itself repeatedly until a specific condition is satisfy this process is called

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n-1)
print(factorial(5))
```

Types of functions:-

1. Built in functions
2. User defined function
3. Lamda functions

Try!

Built-in functions

we have several types of fun^s based on theme characteristics, parameters and return values.

Built-in functions

Python provides many predefined fun^s like print()

```
print()
input()
len()
type()
range()
int()
str()
list()
dict()
bool()
float()
complex().
```

ex program:-

```
num = [1,2,3,4]
print(max(num)) - 4
print(min(num)) - 10
print(sum(num)) - 4
print(len(num))
print(type(num))
```

O/P:- <class 'list'>

User-defined fun^s

User-defined fun^s

1) fun^n with no arguments and no return type

syntax:-

```
def functionname():
    Statement1
    Statement2
    Statement3
```

ex:-

```
def add_numbers():
    a=10
    b=20
    c=a+b
    print("sum:s:", c)
add_numbers()
```

O/P:- 30

function calling

2) function with arguments and no return type.

```
def functionname (P1,P2.....)
    Statement1
    Statement2
    Statement3
```

ex:-
```
def add_numbers(a,b):
    c=a+b
    print("sum is:",c)
```

add_numbers (10,20) → function calling o/p:- 30

3) function with no arguments and return type.

Syntax:-
```
def functionname ():
    Statement1
    Statement2
    State ment 3
    return value1, value2.
```

ex program:-
```
def add_numbers ():
    a=10
    b=20
    c=a+b
    return c → Return to sum
S = add_numbers()
print("sum is:",S)
```

Stores the result and calls the fun-x name

prints the return value.

o/p:- 30

4) function with arguments and return types.

syntax:-
```
def function name (P1,P2....)
    Statement1
    Statement2
    Statement 3
    return value1, value2
```

ex:-
```
def add_numbers (a,b):
    c=a+b
    return c → return to sum
S = add_numbers (10,20)
Print ("sum is:",S) → Prints the return value
```

o/p → 30

Types of arguments:-

```
def f1 (a,b):
    ------
    ------
    ------
f1 (10,20)
```

Here a,b are the formal arguments where as 10,20 are the actual arguments.

* There are four types of actual arguments in Python they are 1) Positional arguments
  2) Keyword arguments
  3) Default arguments
  4) Variable length arguments.

1) Positional arguments:- these are arguments which are passed to a function in absolute positional order.

* The total number of arguments used in a function should be equivalent to the arguments used in fun-x definition. if they are not equal than the function returns a type error.

ex:-
```
def add (x,y):
    sum=x+y
    print (sum)
add (100,200)
add. (100, 200)
o/p:- 300
o/p:- 300
```

2) Default arguments:- This are the arguments for which are default values are assigned during fun-x declaration. these values are used to only when the value of such arguments are not provided in the function call.

ex:-
```
def add (x=100, y=200)
    sum=x+y
    print (sum)
add (10,20) => 30 o/p
add (10)    => 210 o/p
add ()      => 300 o/p
```

Keyword arguments:- These are the arguments which can be identified by the caller or of the fun-x using their parameter names.

* It also the caller of fun-x to specific only few parameters the arguments and in any order.

ex:-
```
def add (x, y=20, z=30):
    sum = (x+y+z)
    print (sum)
add (10,20,30)
add (10, y=40)
add (10, z=20)
```
O/P:-
60
80
50

Variable length arguments:- These are the arguments whose length is not known before runtime or even during excusion and refund is as variable length arguments.

* we can pass 'n' number of values.

ex:-
```
def fun(*no)
    print(no)
fun(10,20,30,40)
```
O/P:-
(10,20,30,40)

Scope of variable:- all variables in a program may not be Access all locations in that program. They are accessible depends on where you have declared values. variables are divide into two types
1) global variable
2) local variable

Global variable:- The variable which are declared outside of the fun-x's, are known as global variables

* global variables of one module can be used through the module (or) program.

Local variable:- The variable declared within in a fun-x are known as local variables. local variables of one fun-x cannot be accesed outside of the fun-x

ex:-
```
x=100
def my-function():
    y=5
    print("Inside function:")
    print("x (global):", x)
    print("y (local):", y)
my-function()
print("Outside function:")
print("x (global):", x)
```
O/P:-
Inside function:
x (global): 100
y (local): 5
Outside function
x (global): 100

Modules:- Modules are files containing python code that can be defined functions, class, variables that can imported and used in other programs. They help in code organisation and reusability.

Creating a module:- To create a phyton module write the code and save that in a file with (.py) extension.

ex:-
```
def add (x,y):
    return (x+y)
def substract (x,y)
    return (x-y)
```

Import module:- we can import the fun-x and class defined in a modules into a another module by using the import statement.

syntax:- import module.

⇒ This does not important the fun-x (or) class directly. to access the fun-x inside the module the (.) dot operator is used.

eg:- import math e → (Euler's theorem)
print(math.e)

O/P:- 2.71

Importing and also renaming:-

* while importing a module we can also change its name

example:- import math as mt
print(mt.e)

output:- 2.71

Python from import module

we can import specific names from a module without importing a module as a whole.

ex:- from math import sqrt, factorial.
print(sqrt(9))
print(factorial(4))

O/P:- 3
24

Import all names:- from import astric statement the (*) symbol used with the import statement is used to import all the names from a module to a current name space.

syntax:- from module_name import *

ex:- from math import *
print(sqrt(16))
print(factorial(5))

O/P:- 4
120

Variables in module

Variables in module contains functions and also variables like arrays and dictionary

ex:-

```
Person = { name: "Joshna", Age: "25" }
import mymodule
a = mymodule.person["age"]
print(a)
```

O/P:- Joshna

modules

python built in functions:-

1) Import math

ex:-
```
import math
print(math.sqrt(25))
print(math.pi)
print(math.degree)
print(math.radians)
print(math.sin(2))
print(math.cos(3))
print(math.factorial(4))
```

O/P:- 5
3.14

2) import random:-

```
import random
print(random.randint(0,3))
print(random.random())
L = [1, 4, True, 'Python', 27, "Hello"]
print(random.choice(L))
```

O/P:- 3
0.45
True

3) import statistics:-

ex:-
```
num = [2,4,6,8]
import statistics
print(statistics.mean(num))
print(statistics.median(num))
```

O/P:- 5
5

4) import datetime:-

ex:-1
```
import datetime
print(datetime.datetime.now())
```

ex:-2
```
import datetime
print(datetime.date.today())
```

5) import calendar:-
To print a particular month in a year we can follow the following code

```
import calendar
print(calendar.month(2025,7))
```

→ O/P: July 2025
Mo Tu We Th Fr Sa Su
1 2 3 4 5 6
7 8 9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

print total calendar in order to all the import calendar months in a year

```
print(calendar.calendar(2025))
```

O/P:- Printed the year calendar

To print a particular mor

Numpy module

Numpy module Provides fun^x for working with arrays and matematical operations

ex:- import numpy as np

```
a = np.array([1,2,3])
b = np.array([4,5,6])
print(a+b)
print(a*b)
print(np.dot(a,b))
```

O/P:- [5,7,9]

[4, 10, 18]

= 32

lamda functions:-

These are the small fun^x that are defined by using the lamda keyword. they can have an number of arguments, were any one expression

Syntax Lambda Arguments : expression

ex:-1 add = Lambda a,b: a+b

```
print(add(5,3))
```

O/P:- 8

ex:-2 square = Lamda a: a**2

```
print(square(9))
```

O/P:- 81

TOP-Down design:- It is a problem solving approch where it inolves breaking down a complex problems into smaller and more managable sub problems. it is also called as Step-wise approch

(or)

It is a problem solving approch where you start by outlining the highest level task your program must perform, then break each of those task into smaller, more managable subtask and until we reach the level there we can write the actual code.

Steps in top-down design.

1) Start with the big problem
(what should the program do?)
2) Break it into main components.
(divide the main task into subtask.)
3) Refine: each subtask
(keep breaking them down until each one is easy to implement.)
4) Implement functions
(code the smallest task as function)
5) Assemble the program
(Call the functions from the top level function)
(or)

1) Define the main problem. (we can create module called list that provides functions for basic list operation)
2) Break down into sub problems
(identify the major task required to solve the problem)
   1) Add task
   2) View task
   3) Remove task
   4) Exit
And also we can perform the operations like addition() sub(), mul(), div().
3) Breaking down each function into smaller function
(enter task to add by using append method.)
(Enter task to remove by using remove method.)
4) Implementation
(Implement each fun^x using Python with list.Py (write "week 2" list Program here)
5) Testing
test the module to ensure it work as expected.
(run the code).
6) Use the module
(To use the list module simple import it in your Python program. Save the file with another way that is main.py.

ex:- import list
list.run()

Files:

Text files:- this type of files consist of the normal characters terminated by the special characters this special character is called EOL (end of line). In python the new line (\n) is used by default.

(Python takes these fites steps for text file).

- Text file in python can be manupldated by using various method and functions.

- * open a file
- * read a file
- * write a file
- * close file.

* first you can create a file with (.txt)
ex:- Hello, Good morning demo1.txt.

Open a file:- To open a text file in python, use the built in open() method. The open function takes two parameters, file name and mode they are different types of modes included in open() method they are

1) r: read mode is used for open a file for reading default.
2) w: write mode is used for write a file ~~for writing a file~~ add,
3) a: append mode is used for read and ~~write~~
4) r+: this mode is used for read and write
5) w+: This mode is used for write and read
6) a+: this mode is used for read and append

Syntax:- f = open ("filename", "mode")

ex:-
```
f = open ("demo.txt", "r")
print (f.read())
```

Reading from a text file:-
To read a file we can use open() method. in read mode (r).

ex:-
```
With open ("sample.txt", "r") as file.
    content = file.read(5)
    print (content)
```

* you have to read one line give (read line)
read all lines give (read lines)

returns the first 5 characters

ex:- f = open ("demo.txt", "r")
print (f.read (5))

ii) read line:-
we can return one line by using readline() method
ex:- f = open ("demo.txt", "r").
print(f.readline())

a) by calling read line two times, we can read the first 2 lines
ex:- f = open ("demo.txt", "r")
print (f.read line())
print (f.read line())

b) By calling read lines we can read all the lines.
ex:-
f = open("demo.txt", "r")
print (f.readlines())

iii)
By ~~can~~ using r+

by using this mode we can read and write,
ex:- f = open("demo.txt", "r+")
print(f.read())
f.write ("welcome")
f.close()

By using w+:
this mode is used for write and read
ex:- f = open("demo.txt", "w+")
f.write ("python")
print (f.read())

4 append
here the mode is used by using this we can add data or text at the end of line (or) text. ~~at the~~.
ex:- f = open("demo.txt", "a")
f.write ("welcome")
f.close()

writing to a file

To write a file in Python we can use write method and example

```
f=open("demo.txt","w")
f.write("welcome")
f.close()
```

6) Close a file:-

by using close method we can close a file.

```
ex:- f=open("demo.txt","r")
     f.close()
```

String Processing:-

A String is a collection of characters which is enclosed by (' ', " ", ''' ''',).

Creating a string:-

```
S1 = 'Hello'
print(S1)
S2 = "Good Afternoon"
print(S2)
S3 = '''To all'''
print(S3)
```

O/P:- 'Hello'

"Good Afternoon"

'''To all'''

String indexing and slicing:-

Python string starts from 0 which is a positive index and ends with '-1' which is a negative index.

Slicing is used to acess the elements from index to another index

ex:-

```
S = "Hello students"
print(s[0])
print(s[-1])
print(s[-2])
print(s[2:5])
```

O/P:- H
s
t
llo

Spacing also taking index

String Concatination

It is used to combine into one string by using '+' operator

ex:- S_1 = "welcome"
S_2 = "to Python"
print (S_1+S_2)
O/P:- welcome to python

String Reptition:-

Repting a string multple times by using astric (*)

ex:- S_1 = "AI&DS"
print (S*5)
O/P:- "AI&DS"
"AI&DS"
"AI&DS"
"AI&DS"
"AI&PS"

1) Define text file? explain different types of modes like r, w, a, r+, wr with examples?
2) ~~Descrip~~ explain string Procesing & it's operations with example
3) Define exceptition handling? explain different types of errors with examples?

String formatting:-

It is used for inserting values into the string

ex:-
colour = clr1 = "red"
clr2 = "white"
print(f"Rose is {clr1} and moon is {clr2}")
O/P:- Rose is red and
moon is white.

String Searching:-

It returns the index of the substring.

ex:- text = "Hello Python"
Print (text.find ["Python"])
Print (text.find ["n"])

O/P:- 7
12

String replacing:-
replace a substring into another string

ex:-
```
s = ("hello", "python")
s = tuple(s.replace("python", "university"))
print(s)
```
O/P:- "hello", "university"

String spliting
Dividing a string into a list of substrings.

```
S = ("Hello, Python")
S = tuple(S.split(S))
Print(S)
```
O/p "Hello", "Python"

```
text = 'APple, Banana, orange"
fruits = text.split(" ")
Print(fruits)
```
O/P:- ["Apple", "Banana", "orange"]

String joining:-
It is a ~~join~~ combining a list of strings into a single string with a specified separator

ex:- words = ["python", "is", "high level language"]
```
Print(" ".join(words))
print("\n".join(words))
```
O/P:- python is high level language

python
is
high level language.

Some common string procesing methods like
1) uppercase
2) lowercase
3) capitilize
4) Strip

ex:-
```
S = "python programming"
print(S.upper())
print(S.lower())
print(S.startwith("P"))
print(S.end with("g"))
print(S.capitalize())
print(S.title())
print(S.strip())
```

O/P:- PYTHON PROGRAMMING
2) Python programming
True
~~False~~ True
P;P
"Python programming"
"Python programming"

Exception Handling:-
It is a process of managing error that occurs during the program execution.
Programing errors are classified into two types
1) compile time error
2) run time error
run time errors are called as exception. handling run time error are called exception handline. it helps to prevent the issue like file not found, division by zero

Common exception types:-
1) ZeroDivisionError:- It occurs when a number divided by 0
2) NameError:- It occurs when a name is not found. It maybe global (or) local
3) IndentationError:- It occurs when a code is incorrect indentation.
4) SyntaxError:- It occurs when a syntax errors are present in code.
5) TypeError:- It occurs when a variable not after excepted type.
6) IOError:- It occurs when input and out operation fails.
7) FileNotFoundError:- It occurs when a file is not found.
exception handling involves using try, except, else, finally blocks to caught and respond to exceptions.

Syntax:-

```
try:
  // block of code
except exception:
  //block of code
else:
  # code runs if no exception

finally:
  #T Statement
```

Try Block:-

It contains the code that might raise an exception.

Except Block:-

It handles the exception if it occurs in the try block.

Else Block:-

It executes when there is no error in try block.

Finally Block:-

The code that always runs whether or not an exception error try block will generate an exception.

ex:-1

```
try:
  print(x)
except:
  print("An exception occured")
```

O/P:- "An exception occured"

Since 'x' is not defined, an exception occured. except runs and print an error occured.

ex2:-

```
try:
  print(x)
except NameError:
  print("variable x is not defined")
except:
  Print (" Wrong")
```

O/P:- Variable 'x' is not defined.

since 'x' is not defined, python raise a name error.

* If it goes to except NameError block and print variable 'x' is not defined.

ex:3

```
try:
  x=1/0
except zeroDivisionError:
  print("The value is not divided by zero")
```

OIP:- The value is not divided by zero.

* It cannot be divided by zero. Divided by zero is not allowed so division error occurs on it print the value is not divided by zero.

ex:-4

```
try:
  print("Hello")
except:
  print("Greetings from Hello")
else
  print("say bye to all")
```

OIP:- Hello
Say bye to all.

* There are is no error in print("Hello"). so except block is skipped.
* else blocks rans and print ("Say bye to all").

ex:-5

```
try:
  print(x)
except:
  print("something went wrong")
else:
  print("Nothing went wrong")
finally:
  print("This code is executed")
```

OIP:- Something went wrong
This code is executed.

* print x will cause any error because x is not defined so the except block runs.
* The else Block is skipped because an error occured in try block.
* The finally block always runs, so it prints the code is executed

ex:6

```
try
  x = 910
except ZeroDivisionError:
      print("It cannot be divided by zero")
else:
      print("Division succesful")
finally
   print("this always runs")
```

O/P:- It cannot be divided by zero

This alway runs.

* x = 910 will cause a '0' zero division error so and except block runs as it cannot be divided by zero.
* The else block is skipped because an error occured in try block.
* The finally block stills runs and print this always runs.

Unit-V

Introduction to object oriented Programming:-

OOPS

The object oriented programming it is used for creating real time projects like: 1) online library management system
2) online Ticket booking
3) online shopping.

Class:- A class is like a blueprint or a template for creating objects.

* In groups data (variables) and behaviour (function/method) together.
* A class is a logical entity that has some specific attributes and methods.
* class is a blueprint for creating objects.
* class is a model for creating objects.

Class declaration:-

Syntax:- class <classname>:
    variable declaration
    method declaration

Variable declaration:- We can declare the variable in two ways
1) Public
2) Private

1) Public:- Here by default the variable is public

Syntax:- Public variable Name = value:

ex:- x=10

2) private:

Syntax:- private __ variable name = value:

ex:- __x=100

Method declaration:-

Syntax:- def methodname:
    statements

A method is just like a function, but it belongs to a class and used to perform operations on the data stored in an object. (An Instance of a Class)

~~Why use method~~

Methods are used to

1) Describe what an object can do (It's behaviour)
2) Access data or change the objects data
3) keep code organised and reusability.

Object

An object is a instance of class with real values.

* It is an entity that existing in the real world.
* It is a Physical entity.

Syntax:-

```
Objectname = classname:
```

Access data of class by using object

Syntax:-

```
Objectname.variable
Object name . method
```

We can access data of class by using class name.

Syntax:-

```
classname. variable
Classname.:method
```

Example:-

```
class Person:
    age = 24
    name = "Joshna"
x = person()
print (x.age)
print (x.name)
print (Person.age)
```

O/P:-

```
24
Joshna
24
```

Example for Accessing data of class using object and class name

Method:- A method is a function that is associate with an Object (that is inside a class).

* Constructor (__init__ method)
* Every class has a special function called __init__ is known as Constructor.

* It is automatically called when an object is created.
* It initialisize the date for the object.
* It uses the self parameter to refer to the current instanct of the class.
* The first parameter of every method in a class must be self.

example program:-

```
Class person:
    def __init__(self,name,age):
        self.name = name
        self.age = age

    def decide(self):
        if self.age>=18:
            print(f"{self.name} is eligible
                for vote")
        else:
            print(f"{self.name} is not
                eligible for vote")

# objects creating & calling methods
X1 = Person('Joshna', 36)
X2 = person('Janu', 18)
X1.decide()
X2.decide()
```

output:- Joshna is eligible for vote
Janu is eligible for vote.

Example program to count number of objects in a class

ex:-

```
class Person:
    count = 0
    def __init__(self):        # it increases the class
        person.count = 1 →     variable count by 1

P1 = person()
print(P1.count)
P2 = person()
P3 = person()
print("The no.of objects =", Person.count)
```

O/P:- 1
The no.of objects = 3

Inheritance :-

It is a process of creating a new class from already existsing class (or)

Inheritance is the capability of one class to acquire properties and characterstics from another class.

* The existing class base class / superclass / parent class is also called

Base
superclass
parent

* The newly created class is cade as child class / subclass / derived class.

Syntax :-

```
class <base class>:
    ==== / Logic(or) code

class <derived class>(<base class>):
    ==== / Logic(or) code.
```

attribute / functions.

Base class (or) Super class :- to give resources to

It is a class which is ready another class

Derived class (or) Sub class :- to take resources from

It is a class which is ready another class.

* The Sub class inheritance data and behaviour (method)

taking

from the superclass.

Advantages :-

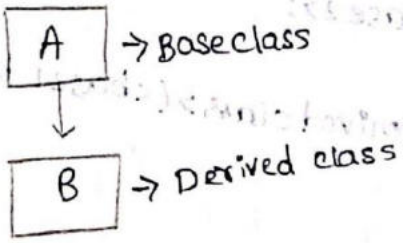
1. Code reusability
2. code ~~extensibility~~ extenstion
3. Time saving

Types of inher

Types of inheritance

1) single inheritance
2) Multilevel inheritance
3) Multiply inheritance
4) ~~Po herihoo hierphical~~
4) hiearay child inheritance
5) Hybrid inheritance

single inheritance:- It enables a derived class to inheritance property from a single parent class, thus enabling code reusbality and the addition of new features to existing code.



syntax:-

```
Class<base class>:
    ---------
Class <derived class>(<baseclass>):
```

example:-

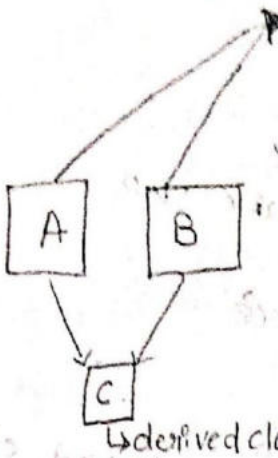
```
Class parent:
    def func1(self):
        print("This function is in parent class")
class child(parent):
    def func2(self):
        print("This function is in child class")
object = child()
object.fun1()
object.fun2()
```

O/P:-

```
This function is in parent class.
This function is in child class.
```

Multiply inheritance:-

When a class can be derived from more than one base class this type of inheritance is called multiply inheritance all of the features of the base class are inheritaned to the derived class.



Syntax:

```
class <base1>:
  .....
Class <base2>:
  .....
class <derived class>(<base1, base>):
```

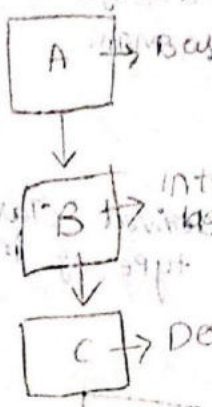
example:

```
class Mother:
    mothername = "SITA"
    def mother(self):
        print(self.mothername)

class Father:
    fathername = "RAM"
    def father(self):
        print(self.fathername)
class Son (Mother, Father):
    def parents(self):
        print("Father:", self.fathername)
        print("Mother:", self.mothername)

s1 = Son()
s1.fathername = "RAM"
s1.Mothername = "SITA"
s1.parents()
```

O/P:

```
Father: RAM
Mother: SITA
```

Multi level inheritance:- In multilevel inheritance, features of base class and the derived class are further inherited into the new derived class. this is similar to the relationship representing a child and grand-father.



Syntax:

```
class <base class>:
    ..... .....
class <derived class1> (<base class>):
    ..... .....
class <derived class> (<derived class1>):
```

example:-

```
class Grandfather:
    def __init__(self, grandfathername):
        self.grandfathername = grandfathername
class Father(Grandfather):
    def __init__(self, fathername, grandfathername):
        self.fathername = fathername
        Grandfather.__init__(self, grandfathername)
class Son(Father):
    def __init__(self, sonname, fathername, grandfathername):
        self.sonname = sonname
        Father.__init__(self, fathername, grandfathername)
    def print_name(self):
        print('Grandfathername:', grandfathername)
        print('Fathername:', fathername)
        print('sonname:', sonname)
s1 = Son('Ramu', 'Raju', 'Mani')
print(s1.grandfathername)
s1.print_name()
```

o/p:- Ramu

grand father name: Ramu

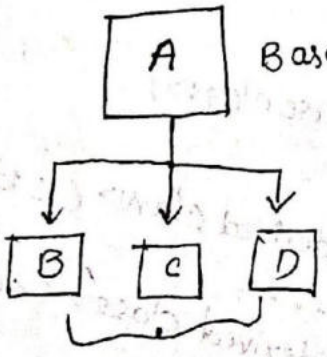
father name: Raju

son name ; Mani

4) Hierarchical

hierarchical inheritance:-

when more than one derived class are created from a singly base class this type of inheritance is called hierarchial class.



syntax:- class <base class>:

---

class <derived class>(<base class>):

class <derived class 2>(<base class>):

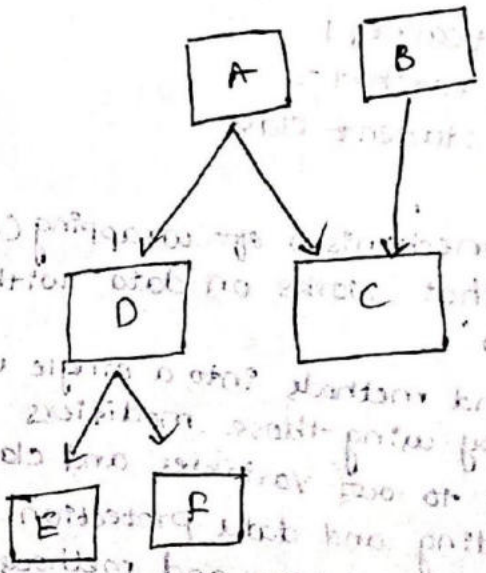
example:-

```
class parent:
    def func1(self):
        print("this function is in parent class")
class child1(parent):
    def func2(self):
        print("This function is in child1")
class child2(parent):
    def func3(self):
        print("this function is in child 2")
object1 = child1()
object2 = child2()
```

```
obj1.func1()
obj1.func2()
obj2.func1()
obj2.func3()
```

O/P:-
```
This function is in parent class
This function is in child1
This function is in parent class
This function is in child2
```

Hybrid inheritance :- It consisting of multiply types of inheritance is called Hybrid inheritance.



Syntax:
```
class <base class 1>:
    ----
    ---
class <base class 2>:
    ----
    ---
class <derived class>(<base class 1>, base class 2):
```

example:-
```
class school:
    def fun(self):
        print("In the school class")
class teacher1:
    def fun2(self):
        print("In a teacher1")
```

```
Class teacher2:
    def fun3(self):
        print("In a teacher2")

Class student(teacher1, teacher2):
    def fun4(self):
        print("In a student class")

obj = student()
obj.fun1()
obj.fun2()
obj.fun3()
obj.fun4()
```

o/p:- In the school class
In a teacher1
In a teacher2
In a student class

Encapsulation :-

Encapsulation is a mechanism of wrapping (combing) data and methods that works on data with in one unit that is class (or).

wrapping of variables and methods into a single unit access specifier (or) modifier by using these modifiers we can provide data security to our variables and class.

* It provides data hiding and data protection.
* It restricts acess to variables and methods directly.
* It helps achieve data hiding and maintability in code.
* encapsulation is implemented using acess modifiers such as public, private and protected.

Public:- The public members are accessible from any where both inside and outside the class these are default members in Python.

Syntax:-

Variable

ex:-

```
Class demo:
    x=100
    def show(self):
        print("Hello")
```

```
a=demo()
a.show()
print(a.x)
```

o/p:- Hello
100

ex-2:-

```
class public:
    def __init__(self):
        self.name="Joshna"
    def display_name(self)
        print(self.name)

obi= public()
obi.display_name()
print(obi.name)
```

o/p:- Joshna
Joshna.

private:- Private members accesable only with in the class.
It is indicated by (--).

syntax:-

```
--var
```

ex:-

```
class demo:
    --x=100
    def show(self):
        print(self.--x)

a=demo
a.show()
```

o/p:- 100

when both variable & function are provide Private.

ex:-

```
class demo:
    --x=100
    def --display(self):
        print(self--a)
    def show(self):
        self.--display()
```

e.demo()
e.show()

O/P: 100

Protected:- Protected members are added in class, it's sub class. It is indicate by a (-).

Syntax:-

_var

Example:-

```
class demo
    _x = 100
def show(self):
    print (self._x)
```

```
class demo:
    def __init__(self,name):
        self.name = name
class demo1(demo):
    def display(self):
        print (self._name)
a = demo1("Joshna")
a.display()
```

O/P:- Joshna.

Polymorphism

It refers to having multiple forms. Implementing method into single unit. It is a foundation and also entity like function, method or operators to behave differently based the type of data.

Polymorphism

Polymorphism in Built-in functions:

```
print (len("Hello"))
print (len [1,2,3])
print (max (1,3,2))
print (max ("a","c","z"))
```

O/P:- 5
3
3
z

Polymorphism in function

```
def (add (a,b):
    return (a+b)
print (add (3,4))
print (add ("Hello", "world"))
print (add [1,2], [3,4])
```

O/P:- 7
Helloworld
[4,6]

Operator Overloading

In Operator overloading the operators like '+' behave polymorically performing addition, concentation are merging based on data type.

example:-

```
Print ('5'+10)
print ("Hello"+"Python")
Print ([1,2]+[3,4])
```

O/P:- 15
HelloPython
[4,6]

The Polymorphism can be classified into two types.

" we cannot perform '+' operator on obj
If we can we can create method (self, other)

* Method overloading
* method overwritting
* Method overloading :-

It happens when a class has two or more methods with the same method name but different parameters is called method overloading.

ex:-

```
Class M:
def S(self,a):
    print(a)
def S(self,a,b):
    print(a+b)
def S(self,name,age,a):
    print("name is:",name)
    print("age is:",age)
    print(a)
Obj=M()
Obj.S('Ramu',20,10)
```

O/P:-

```
a
a+b
name is: Ramu
age is: 20
10
```

Method overwriting

It having two methods with same method name with same parameters, one of the method name is in Parent class and other method is in child class.

* Method ex:-

```
class Animal :
    def speak (self):
        return "some sound"
class Dog (Animal ):
    def speak (self):
        return "Bark"
class cat (Animal):
    def speak (self):
        return "Meow"

def animal_speak(animal):
    print (animal.speak())

dog = Dog()
cat = Cat()
animal_speak (Dog)
animal_speak (Cat)
```

O/P:- "Bark"
"Meow"