Introduction to Python:-

History of Python:-

→ Python was developed by Guido van Russum in the year 1989
→ Python was developed much before java but due to unavailabile-ty of packages the programming used much latter,
→ Python was released officially in February 1991.
→ Python was categorized under high level programming languages because it supports language.
→ Python uses compiler or interpreter according to need of the program. Python is referred as interpreted language because it will use interpreter over compiler many times.

Features of Python:-

→ The developers have used features from diff programming language to develope the python programming language,
→ function oriented approach is taken from C programming language.
→ Object oriented programming approach is taken from c++.
→ Scripting approach is taken from perl & shell scripting.
→ Modular programming approach is taken from module-3

Where we can use python?

Due to the availability of various packages we can use Python by embadding it to diff technologies. Listed below

1. For developing desktop applications.
2. For developing web applications.
3. For network applications.
4. For Database applications.
5. IoT applications.
6. machine learning models.
7. Data analysis and Preparation models.
8. For developing games.

Date
Page

Programming Properties of Python' -

→ The following properties make python programming widely used over other programming languages & their.

1. Simple & easy to learn.

i). Python programming is called simple because they used only 30+ keywords which makes the program easy to learn.

ii) Like other programming languages python programming doesn't need huge line of code it only needs one or two line to do basic operations which makes the program readable & simple.

2. Open source & freeware -

i) Python prog. is an open source software so everyone can use it without any licence.

ii) It is a freeware that makes the source code can be shared & open to all for modification.

3. High level programming language) - 01-04-01-2023

i) Python is called high level prog. language because it uses keyword & operations to write the programs.

ii) The low level activities like memory management, seceurity etc. will be taken care by the system.

4. Dynamically typed! -

→ In python variables are dynamically typed ie. the value will decide the data type of the variable when it is assigned

→ In python prog. the data type of the variable is not fixed

5. Platform independent' -

→ Python prog. support platform independence that means once a prog. is written that can be executed in any of the platforms

→ We only need to change the PVM according to the platform.

6. Portable' -

→ Python prog. are portable because they can be easily transferred from one system to another.

→ The code will generate the same result on all the platforms.

7. Both procedural & object oriented! -
→ Python programming follows procedure oriented as well as object oriented approach that means we can write programes by using functions or using class & object.
→ By using both the approaches we can get the advantages like security, reusability, maintainbility, user modification allowing.

8. Interpreted! -
→ Pythons programes are interpreted because it uses interpreter to convert the code into machine level language.

9. Extentiable! -
→ Python can be extended with diff programming languages that provides the following advantages.
i) It improves the performance of the code.
ii) The code in other languages can be reused by the python
iii) The extension can be done by java. net, html, etc.

10. Embadded! -
→ We can use others languages by combining it with python that makes python programes embadded.

11. Extensive library -
→ Python provides a wide range of library containing different modules & packages that can be used for various applications.
→ The diff library modules are Numpy, Pandas SCITECH, MATPLOT, LIB

Limitations of Python! -
Python has the following limitations:-
→ Performance wise python alone is not upto the mark.
→ Python doesn't provide any modules for mobile applications

Parts of Python! -
A Python program can be divided into the following.
i) Identifiers iii) operators iv) Datatypes.
ii) Keywords iv) Literals.

i) Identifiers:-
→ Identifiers are the names given to the values to identify them in a programes.
→ The identifiers can be any named except the keywords.
→ The different identifiers are functions name, variables name, class name etc.
eg:- A=10, where A is the identifier.

ii) Keywords:-
→ The keywords are the words that has fixed meaning in a programming language.
→ we have 35 keywords in python. In total making it 33.

iii) Operators:-
→ These are the symbols used for operating the values & they are i) Arithmetic operators:-
The following arithmetic operators are provided by python & they are
a) + c) * e) %
b) - d) / f) // (Floor division):- The operator returns the floor value of the division.
→ This division can be used with float as well as integer calculation. If the operands are int then it will return int as usual if the operand are float then it will return float as usual.

g) ** (Exponent)
→ This operator returns the exponent or the power of one number raised to another number.
→ This can be used as 2**4 (4 → power, 2 → base)

ii) Relational operators:- Relational operators in python are <, ≤, >, ≥

iii) Logical operators

Date 6/01/2023
Page

iii) Logical operators:-
→ Python provides the following logical operators that are used to compare conditions.
→ The diff. logical operators provided by python are:
* and
* or
* not

iv) Assignment operators:-
→ This operator assigns values to a variable either directly or from another variable.
→ Ex:- a=10
b=a

v) Bitwise operators
→ we have the following bitwise operators that work on the bit values of the operand.
→ The different bitwise operators are.
* and (&)
* or (|)
* Ex-or (^)
* negation (~)
* left shift (<<)
* right shift (>>)

vi) Special Operators:-
→ The special operators are limited to python programming only.
→ The different special operators are i) identity operator
ii) membership operator

* Identity operator:-
→ This operator is used for address comparision.
→ we have to identity operators i) is
ii) is not

Ex:- Use identity operator for comparing two different variable

→ a = 10
b = 10
Print(a is b) //True
Print(a is not b) //False.

* Membership Operator

→ This operator is used to check whether the given element/object is present in the collection of objects/elements.

→ we have two different membership operators
i) in
ii) not in

Ex:- Use membership operator for checking the presence of an element in a collection

Ex:- x = "Hello"
Print('b' in x) //False
Print('H' in x) //True.
Print('H' not in x) //False

iv) Literals:-

→ Literals are the values that never change.
→ literals are the inputs for the variables.
→ we have different types of literals according to the values they represent & they are i) Integer literals ii) Floating point iii) Character literal iv) Boolean literal v) String literal

i) Integer literals :- 5, 6, 7
ii) Floating point literals :- 7.5
iii) character literals - 'P'
iv) Boolean literals - True or False
v) String literals = "Pratyush"

Date
Page

* Constant:-
Constants are the variables whose value doesn't change.
→ we can define constants in python using the keyword const. before the variables.
→ This will make the value of the variable fixed during the operation.

Ex:- const a = 10

V) Data types:-
→ Datatypes are representing the types of value provided to a variable.
→ Datatypes are not explicitly provided to the variable in python.
→ Python assignes the datatypes to the variables according to the values assigned.
→ we have the following kinds of datatypes in python

1) int
2) Float
3) Complex
4) Bool
5) Str
6) Bytes
7) bytes array
8) range
9) list
10) tuple
11) set
12) frozenset
13) Dict
14) None

1) int:-
- This represents integere numbers.

2) Float:-
- This represents all floating point numbers.

3) Complex:-
- This represents all the complex numbers in form of (a+ib) → a → real part
ib → imaginary part
Ex:- a = 5 + i6

4) Bool:- Represent the boolean values True/False.
5) String:- It represents all the collection of characters Eg:- "Pratyush".
6) Bytes:- This represents the numbers ranging from 0 to 256.

```
x = [10, 20, 30, 40]
b = bytes(x)
```

7) byte array:- It is the collection of number of bytes represented in the form of array.

```
x = [10, 20, 30, 40]
b = bytes(x)
b1 = bytes(x)
Y = [b, b1]
```

Range:-
→ range represents a collection of values between a minimum & maximum value.
→ The range will always have one minimum & one maximum value & step.
Eg:- range(0, 100)
range(0, 100, 2)

Dt-07/01/2023

→ Out of the three things minimum & step are optional. Everytime we need to provide the maximum value i.e. compulsory.
→ Default value for minimum is zero & default value for step is 1

Form 1:- range(10)
→ generate numbers from 0 to 9.

Form 2:- range(10, 20)
generates numbers from 10 to 19.

Form 3:- range(10, 20, 2)
2 means increment value
10, 12, 14, 16, 18

List:-

→ list is a collection of different kinds of data in a single entity.

eg:- 1) List1 = [10, 10.5, 'durga', True, 10]
print(List1)

tuple:-

- tuple is also a collection of elements where we can collect different types of values.
- The tuple can be represented as ( )

e.g:- tuple = (10, 10.5, 'data')
print(tuple)

Set:-

→ set is also a group of elements without duplicate elements

s = {10, 20, 'data', 10.5}
print(s)

Frozen set:-

→ Frozen set is equivalent to set only difference is it is immutable

Date 07/01/2023
Page

e.g:- s={10, 20, 20.5, 'data'}
s1 = frozenset(s)
Print(s)
Print(s1)

dict:-
→ dictionaries also a collection of elements but represented as pairs of key & value.
eg: d={101:'durga', 102:'ravi', 103:'shiva'}
key value.

None:-
→ None is the return type for nothing.
→ It is equivalent to the null value in java.
eg:-
def m1();
a=10
Print(m1())
None.

Escape Sequence / character:-
→ These are the special characters or string literals used for formating the output.
→ Each escape sequence has a separate meaning & use.
→ The different escape sequences are
1) \n ⇒ New line
2) \t → Horizontal tab
3) \r → Carriage Return
4) \b → BackSpace
5) \f → Form feed
6) \v → vertical tab
7) \' → Single quote
8) \" → Double quote
9) \\ → black slash Symbol

Type casting:-

→ Converting one data type into another is called type casting or type conversion.

→ Typecasting are of 2 types: 1) Explicit
2) Implicit - (lower to higher)

→ Explicit type casting in python can be done using the following functions:

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

1) int():-

→ we can use this function to convert other type functi to int.

eg:- >>> int(123.987)
2) 123

2. float():-

→ Using this function we can convert other type functio to float

eg:- >>> float(10)
10.0

3. Note:-

- Using int() & float() function we cannot convert strings into int or float because the characters use UTF code instead of ASCII for representing in system.

3. complex():-
This function is used to convert values of other types into complex type:-

* form1:- complex(x)
- In this function x is converted into real part x & imaginary part with 0.
e.g:- complex(10) → 10+0j

* form2:- complex(x, y)
- In this the x will be converted as real part & y will be converted into imaginary part.
eg: complex(10,-2) → 10-2j
complex(True, false) → 1+0j

01-10/01/2023

4. bool():-

→ This function is used to convert other types into boolean type.
→ The use of bool function can be done in the following ways.
e.g

bool(1) → True
bool(10.5) → True
bool(0+1.5j) → True
bool("false") → True

Note:-
bool function returns false for 0 & its equivalent values. For all other values it will generate true.

5. str() :-

→ This function is use to convert values from other type into string type.

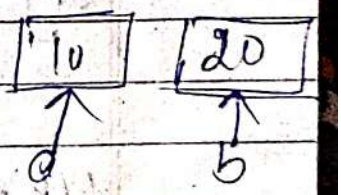
→
e.g: str(10) ⇒ '10'
str(10.5) ⇒ '10.5'
str(1+5j) ⇒ '1+5j'
str(True) ⇒ 'True'

Data types vs immutability :-

→ Immutability represents the concept of utilizing same memory location for different variables.
→ Immutability also represents binding of objects with values. which cannot be modified, If modification is done then a new object will be created.
→ In python defining two different objects with same value will create only one object with two object references if the values are same, otherwise two different references with two different objects will be created.
e.g:-

i) a=10
b=10

ii) a=10.
b=20



→ If objects are not immutable then any change to one object will reflect in other objects also. to prevent that immutability is supported in python.

Const-

Input and output in python:-

→ Python supports dynamic input from keyboard using two different functions. i) raw_input()
ii) input()

i) raw_input():-

→ This function takes input from from the user in form of string.
→ The function can be used in the following way.
eg:-

```
x = raw_input("Enter a value")
Print(x)
```

O/P ⇒ Enter the value: 10
→ '10'.

ii) input():-

→ This function also takes input from the key-board directly in the user required format.
→ The input function can be used in the following way
e.g:

```
x = input("Enter a value")
Print(x)
```

O/P:- Enter the value 10
10

Note:-
On python 3 only input function is available which takes the input from the key board in string format & we have to type cast it explicitly into user required format.

eg:- x = int(input("Enter a value"))
print(x) // Enter a value: 10
10

* Output in python:-

→ Python provide print function & various format specifiers/Escape sequences for getting the output

→ The print function will be used in various format to get different kinds of output
→ The different uses of print function are
i) Form 1
Print() without any argument it will print a new line character
ii) Form 2
Print with string
Print("Hello World")
Print("Hello" * 10)
iii) Form 3
Print with multiple variables
eg a, b, c = 10, 20, 30
Print("The values are:", a, b, c)
Output The values are 10 20 30

* Using separation attribute

a, b, c = 10, 20, 30
print(a, b, c, sep = ',')
O) 10, 20, 30

Form 4:-
Print() with end attribute

Print("Hello")
Print("Durga")
Print("Soft")

1. Print("Hello", end=' ')
2. Print("Durga", end=' ')
3. Print("Soft")

Form-5:-
Print with object:-
→ This print() function can directly print all the elements of a collection object like list, tuple set etc

e.g:- l=[10,20,30,40]
t=(10,20,30,40)
Print(l)
Print(t)

form 6:-
Print() with string are variable list:-

By Print() function we'll print the combination of string with other variable by using a separator

→ The comma will use as concatination of Strings & Separator bw Strings & other datatypes

Eg:

```
s= "Durga"
a=48
S1="Java"
S2="Python"
Print("Hello",s,"your age is",a)
Print("You are teaching",S1,"and",S2)
```

Output

1) Durga your age is 48
2) you are teaching Java and Python

Form 7)-

Print() function with formating.

→ This print function will take different format specifiers for printing different types of values.

%i ⟶ int
%d ⟶ int
%f ⟶ float
%s ⟶ string

Eg:-

```
1) a=10
2) b=20
3) c=30
4) Print("a value is %i"%a)
5) Print("b value is %d and c value is %d"%(b,c))
```

Output:-

1) a value is 10
2) b value is 20 and c value is 30.

8) form 8:-

Print() with replacement operator:-

→ This function will print the output and the specified replacement

eg:- name = "Durga"
salary = 10000
gf = "Sunny".
Print("Hello %s your salary is %d and your friend %s is waiting". format(name, salary, gf))

Print("Hello {x} your salary is {y}. and your friend {z} is waiting". Format(x = name, y = salary, z = gf))

Output

1) Hello Durga your salary is 10,000 and your friend Sunny is waiting.
2) Hello Durga your salary is 10000 and your friend is waiting.

* Flow control:-

→ These are the structures used to control the flow of the program.
→ we have three different statements to control the flow & they are: 1) conditional statements
2) Transfer or Jump statements
3) Iterative or Looping statements

1) Conditional statement:-

→ These statements are used to check the condition which are categorized into three different types

i) if
ii) if-else
iii) if-elif-else

i) if:-

This structure controls the program using the condition & the statements which will execute when the condition becomes true.

Syntax:-

if condition: statement

(or)

if condition:
    statement-1
    statement-2
    statement-3

if condition is true then statements will be executed

eg:

```
1) name = input("Enter name:")
2) if name == "durga":
3)     Print("Hello Durga Good Morning")
4) Print("How are you!!!")
5)
```

Output

7) Enter name: Durga
8) Hello Durga Good Morning.
9) How are you

If-else:-

→ This construcrure controls the statements accordingly to the condition & its results.

Eg: Syntax:-

```
If condition:
    Action-1
else:
    Action-2
```

If condition is true then Action-1 will be executed otherwise Action-2 will be executed.

Eg:

```
n) name=input("Enter Name")
If name=="durga":
    Print("Hello Durga Good Morning")
else:
    Print("Hello Guest Good Morning")
Print("How are you!!!")
```

O/P:-

If-elif-else:-

→ This structure is used to check multiple condition.

Syntax:-

```
If condition 1:
    Action 1
elif condition 2:
    Action 2
```

elif condition3:
    Action3
elif condition4:
    Action4
    ..
    ..
else:
    Default Action.

Eg:
1) brand=input("Enter your favorite Brand")
    if brand == "RC":
        Print("It is children brand")
    elif brand == "KF":
        Print("It is not that much krew")
    elif brand = "FO":
        Print("Buy one get one free")
    else:
        Print("Other Brands are not recommended")

O/P
Enter your favorite Brand: RC
It is Children Brand

II) Iterative Statement
These statements are used to execute a group of statements multiple times.
→ Executing group of statement multiple times can be done in 2 different ways.
1) using for loop
2) using while loop

i) Using for loop' -

→ This loop will be used for operating the elements in sequences, like strings, list, tuple etc.
→ The for loop can be used in the following way.

Syntax:

```
for x in sequence:
    body.
```

* WAP to print a characters present in the given string

```
S= input('Pratyush')
for i in S:
    Print(i)
```

```
S: input('Pratyush')
For i in S:
    Print(i)
    i= i+1
```

ii) Using while loop' -

→ While loop is also used to print or execute multiple statements iteratively until a condition becomes false.
→ While loop is preferred when we want to use some condition as stop condition.

Syntax:

```
while condition:
    body
```

* WAP to print from 1 to 10 using while loop

```
i= input i= 1
while      while i<=10:
               Print(i)
```

* WAP to take input from the user & print it until the user enters 0.

~~x = input(x = int(input("Enter the number:"))~~
~~while x != 0:~~
~~print(x)~~

~~for~~
→ x = int(input("Enter the numbers:"))
while x != 0:
print(x)
n = int(input("Enter a no:"))

* WAP to print all the even numbers between 1 to 50

* ~~For loop~~
~~for i~~ ~~i = int~~
~~i = int~~
For x in range(1, 50):
~~print(x)~~

For loop"
For x in range(1, 51):
if x % 2 == 0:
Print(x)

while loop
~~x = 0~~ x = 1
~~while(~~ while x
~~x = 1~~
while x <= 50:
if x % 2 == 0
print(x)
x = x + 1

WAP to print the sum of even & odd numbers separately in between 1 to 100.

```
S1=0, S2=0
For x in range(1,101):
    if x%2==0:
        S1 = S1+x
    elif x%2!=0: else
        S2=S2+x
Print("The sum of even no is:",S1)
Print("The sum of odd no. is:",S2)
```

While loop

```
x=1, S1=0, S2=0
while x<=100:
    if x%2==0:
        S1=S1+x
    else:
        S2=S2+x
    i=i+1
Print("The sum of even no. is:",S1)
Print("The sum of odd no is:",S2)
```

*Nested loops:-

→ Nested loops are the structures where we will use one loop inside another loop.
→ For nested loop we can combine similar or dissimilar loops.
e.g:-

```
For i in range(4):
    For j in range(4):
        Print("i=",i,"j=",j)
```

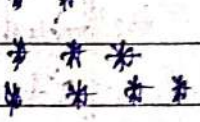
* write a program to display * in right angle Δ form.

```
n=int(input("Enter the value of:"))
for i in range(n):
    for j in range(i):
        Print('*')
    Print(\n)
```

```
n= int(input("Enter the value of n:")
for i in range(1, n+1):
    for j in range(1, i+1):
        Print("*", end=" ")
    Print()
```

O/P

enter the value of n: 4



```
n= int(input("Enter the value of:")
for i in range(n):
    for j in range(i):
        Print("*", end=" ")
    Print()
```

* WAP to print the following pattern:

```
n= int(input("Enter the value of n:")
for i in range(n):
    K=i
    for j in range(i):
        Print('K', end=" ")
    K=K-1
    Print()
```

1
2 1
3 2 1
4 3 2 1
5 4 3 2 1

#WAP to print the following pattern.

```
    1
  1 2 1
1 2 3 2 1
```

Infinite loop

→ If the loop is not satisfying a condition to stop then it will result in infinite loop.

→ When the stop condition for while loop goes wrong it may lead to infinite loops.

eg:
```
i=0
while True:
    i=i+1;
    Print("Hello",i)
```

Transfer or Jump statements:-

1) break:-

→ This statement is used to stop the execution of currently executing loop where the break statement is present.

→ e.g.
```
for i in range(10):
    if i==7:
        Print("Processing is enough... Plz break")
        break
    Print(i)
```

O/P
0
1
2
3
4
5
6
Processing is enough... plz break

2) Continue! –
→ This statement is used to skip the execution of current iteration.
→ It will also be used to continue into the next iteration by shifting the control from previous to next
e.g

```
For i in range(10):
    if i == 7:
        continue
    Print(i)
```

O/P
0
1
2
3
4
5
6
8
9

3. Pass! –
→ The pass statement is used to do nothing inside a loop.
→ The pass statement will be used to define empty blocks
e.g

```
if True: ✗
    SyntaxError: unexpected EOF while parsing.
if True: Pass ✓
    ⇒ valid
```

Eg:2 If True: pass

else:
print()

→In the above example it is a blanck block due to the use of pass statement

Eg:2
def m1():
SyntaxError: unexpected EOF while parsing.
def m1(): pass

Dt-13/01/2023

Strings:-

→ String in python are represented by the data type 'str' & which can be typecasted using the function 'str()'.
→ Strings represent a collection of characters that can be enclosed within single quotes or double quotes.

Syntax
s='durga'
s="durga".

→ Strings can be also created using multiple lines definations using three single quotes or double quotes at the begining and at the ending of lines

Syntax:
s: "durga
software
solution".
→ For accessing the character of a string we can use two different way:
1) using indexes
2) using slice operator

By using index

→ It is similar to
→ we can use the indexes for getting the elements or characters of a string.
→ Python supports 'tve' & -ve index.
→ +ve index moves the string left to right & the -ve index moves the string from right to left.

s = 'durga'
s[0]
→ 'd'
s[4]
→ 'a'
s[-1]
⇒ 'a'

Using Slice operator:-
→ we can also use the slice operator to get the characters of a string.
→ The slice operator will returns the substring of a original string.
→ we can use the slice operator as
Syntax:- s[beginindex : endindex : step]

eg.
→ begin index: This represents the index from where the slicing will start.
→ endindex: This represents the index where the slicing will stop which is excluded.
→ Step represents the incremented value where the consideration of next value is required.

e.g:- S = "Learning Python is very very easy!!!"

```
S[1:7:1]
'earnin'
S[1:7]
'earnin'
```

Behaviour of Slice operator:-

→ The default value for begin index is 0 & endindex is slice-1 & step is 1.
→ We can give the step value as either +ve or -ve if it is +ve then forward movement will be done if it is -ve backward movement will be done.

Operators for String:-

→ We can use two mathematical operators for String i) + for concatenation
ii) * for repetition.

```
Print ("durga" + "soft") #durgasoft
Print ("durga" * 2) #durgadurga
```

Note
1. To use + operator for strings, compulsory both arguments should be str type.
2. The repetition operator replaces a loop for

Printing the same string multiple times:-

Finding length of a string:-

→ Length of a string can be found out using built in function. [len()]

Eg:
```
s='durga'
Print(len(s)) #5
```
→ we can also find the length of a string using loop

eg:- s = 'Pratyush'
```
i=0,
while i in s:
    Print(i)
    i=i+1
```

Checking Membership:-

→ we can check whether a character is present in the string or not using the membership operators.

eg:1)
```
s='durga'
Print('d' in s)#True
Print('c' in s)# false
```
eg:2
```
s='Pratyush'
Print('d not in s) # True
Print('P not in s) # false
```
Note:-

→ we can use the membership operator for checking multiple characters or substring in a given string.

→ In this membership the sequence of characters will be matched with the given string.

Comparision of strings:-

→ The relational operators can be used to compare strings in python.

→ The different operators that can be used are
<, <=, >, >=, ==, !=

→ E.g

```
s1= input("Enter your 1st String:")
s2= input("Enter your 2nd string:")
if s1==s2:
    Print("Both Strings are equal")
elif s1>s2:
    Print("1st string is greater than 2nd String")
else
    Print("1st string is less than 2nd String")
```

O/P

Enter your 1st string: durga
Enter your 2nd String: durga
Both strings are equal

Removing space from the string:-

- For removing space from string, we have 3 methods.

1) rstrip() => To remove space from right side
2) lstrip() => To remove space from left side
3) strip() => To remove spaces both sides.

eg:

```
city= input("Enter your city Name")
Scity=city.strip()
if Scity=="Hydrabad":
    Print("Hello' Hyderabad... Howdy")
else:
    Print("your entered city is invalid")
```

Note:- This function will remove the space from begening or ending of the string only.

Finding Substring:-

We have 4 different methods for finding substrings

→ The different methods are i) Find()
ii) index()
iii) rfind()
iv) rindex()

Dt:-17/01/2023

1) Find()

→ This method is used to find the first occurence of the given substring.
→ In order to use defined method the syntax is
→ S.find(substring)

eg.1- S="Learning python is very easy".
print(S.find("Python")) #9

Note:-
If the substring is not present then '-1' will be returned.

2. index()' -

→ This method is used to find the substring similar to find() method. The only difference is if the substring is not present instead of '-1' we will get value error.

eg:-

```
S=input("Enter main string:")
subs=input("Enter sub string:")
try:
    n=S.index(subs)
except ValueError:
    print("substring not found")
else:
    print("substring found")
```

3. rfind():-
→ rfind() method works similar to find, the movement is done from backwards.
→ The calculation of indexes will be done from right to left

eg:- S="Learning Python is very easy"
Print(S.rfind("Python"))
#9

Note:-
The rfind() method will start the calculation of indexes from the position of the substring upto the begining of the string in backward direction but find method will calculate the indexes starting from the begining of the string upto the starting of the substring

4. rindex():-
→ This method is similar to index method the only difference is calculation will be done in backwards.

Counting substrings in a given string:-
→ Strings provide a method count that can be used to count the no. of occurences of a substring in a given string.

Two types of syntaxes are there.
i) S.count(substring) ⇒ It will search throughout the string
ii) S.count(substring, begin, end) ⇒ It will search from begin index to end-1 index

Eg:-
s="abcabcabcabcaddo"
Print(s.count("a"))
Print(s.count("ab"))
Print(s.count("a",3,7))
Output
6
4
2

Re: replace():-
→ We can replace a part of a given string with a new string using this method.
→ The replace() method will be used in the following way
Syntax:-

s.replace(oldstring, newstring)
Eg:
s="Learning python is very difficult"
s1=s.replace("difficult","easy")
Print(s1)
Output Learning python is very easy.

* String objects are immutable but how replace method modifies the string.
→ String objects are immutable so once created cannot be modified.
→ When we are using replace method a new object reference is created every time without changing the old object that maintains the immutability of strings.

Split():-
→ This method is used to generate a complete string into smaller parts using a separator.
→ The separator will be used by the method to separate the strings into substrings.
→ The separators can be space or special characters in both the string.

Syntax:
L = S.split(separator)

Eg:-
S = "22-02-2018"
L = S.split('-')
For x in L:
    Print(x)

Output
22
02
2018

Join():-
This method will be used to join a group of strings in to a single string using the specified separator.
→ The join method will be used in the following method
Syntax: separator.join(group of strings)
Eg: t = ('sunny', 'bunny', 'chinny')
S = '-'.join(t)
Print(S)
Output:- sunny-bunny-chinny.

Changing Case of string!-

→ Python provides seven different methods to change the case of a string.

1. upper() ⇒ To convert all characters to uppercase.
2. lower() ⇒ to convert all characters to lowercase.
3. swapcase() ⇒ converts all lower case characters to upper case and all upper case character to lower case
4. title() ⇒ To convert all character to title case. i.e. first character in every word should be upper case and all remaining characters should be in lower case
5. Capitalize ⇒ only first character will be converted to upper case and all remaining characters can be converted to lower case.

```
S='learning python is very easy'
Print(S.upper())
Print(S.lower())
Print(S.swapcase())
Print(S.title())
Print(S.capitalize())
```

output

```
LEARNING PYTHON IS VERY EASY
learning python is very easy.
LEARNING PYTHON IS VERY EASY
Learning Python Is Very Easy.
Learning python is very easy.
```

Date
Page

Checking, Starting and Ending of a string:-

→ Python provides two methods for this purpose, they are

1) s.startswith(substring)
~~2. s. startswith~~
2. s. endswith(substring)

eg:-

```
s='learning Python is very easy'
Print(s.startswith('learning'))
Print(s.endswith('learning'))
Print(s.endswith('easy'))
```

Output

True
false
True

Note:-

→ These two methods returns boolean values as result and are case sensitive.

check type of characters present in the string:

The following methods are used to check the type of character.

1. isalnum(): Returns True if all characters are alphanumeric (a to z, A to Z, 0 to 9)
2. isalpha():- Returns True if all characters are only alphabate symbols (a to z, A to Z)
3. isdigit():- Returns True if all characters are only digits (0 to 9)

4. islower(): Returns True if all characters are lower case alphabet symbols
5. isupper(): Returns True if all characters are uppercase alphabet symbols.
6. istitle(): Returns True if string is in title case.
7. isspace(): Returns True if String contains only spaces.

Eg: Print('7869'.isdigit()) # True

Formating the Strings:-

→ We can format the strings in python using replacement operator {} and format() method.

eg:
```
name='durga'
salary=10000
age=48
Print("{}'s salary is {} and his age is {}".format(name,salary,age))
Print("{0}'s salary is {1} and his age is {2}".format(name,salary,age))
Print("{n}'s salary is {s} and his age is {a}".format(name,salary,age))
```

Output

durga's salary is 10000 and his age is 48.
durga's salary is 10000 and his age is 48
durga's salary is 10000 and his age is 48

Q Program to reverse the order of words.

Input: Learning Python is very Easy
Output: Easy very is Python Learning.

```
S= input("Enter some string:")
l = S.split()
ll=[]
i = len(l) -1
while i>=0:
    ll.append(l[i])
    i=i-1
Output = ' '.join(ll)
Print(Output)
```

Output

Enter some string: Learning python is very Easy!!!
Easy !!! very is Python Learning.

Date 19-01-2021

List Data Structure:

→ List data structure is equivalent to an array where we will represent collection of element as single entity
→ List represents a single entity having the following properties
1) Insertion order is preserved ~~to dupli~~
2) Duplicate objects are allowed
3) Heterogeneous objects are allowed
4) Dynamic in nature.
5) Elements will be placed inside square brackets
6) Mutable

Separated by ','
→ list elements are accessed using indexes that normally starts from '0' moves upto no. of elements-1
→ Indexes in list is used to differentiate the duplicate elements.
→ The indexes of a list can be '+ve' or '-ve'.

list [10, "A", "B", 20, 30, 10]

-6	-5	-4	-3	-2	-1
10	A	B	20	30	10
0	1	2	3	4	5

→ list objects are mutable that represents a list which can be modified after creation.

Q WAP to create a list & print the elements.

```
List = eval(input("Enter list:"))
Print(list)
print(type(list))
```

→ The eval() function is used in the above program evaluates all the elements & stores it into a list.

List() function :-
→ This function will create a list out of the collection
→ The collection can be string, range, tuple etc.
Eg:
1.
```
I = list(range(0,10,2))
Print(I)
```

2.
```
S = "dinga"
l = list(S)
Print(l)    # ['d', 'i', 'n', 'g', 'a']
```

Split() function:-
→ It is used to create sub list from a list.
→ It also separates parts of a string and creates a list.
Eg:-
S = "Learning Python is very very easy!!!"
I = S.split()
Print(I)
Print(type(I))
# ['Learning', 'python', 'is', 'very', 'very', 'easy!!!']

Note:
Sometimes we can take list inside another list, such type of lists are called nested lists.
[10, 20, [30, 40]]
→ When we provide a list as an element of another list then it is known as nested list.
→ In order to access the nested list we will use the normal indexing for the nested list with the original list.

Note
If we want to access the elements of inner list then we must take that list into another list variable & access using normal indexing.

Eg: x = [10, 20 [30, 40]]
y = x[2]
Print(y[1])
# 40.

Accessing elements of list:-

→ We can access the list elements by two different way i) By using index.
ii) By using slice operator.

By Using Index:-

```
list=[10,20,30,40]
Print(list[0])==>10
Print(list[-1])==>40
```

By Using Slice operator:-

```
n=[1,2,3,4,5,6,7,8,9,10]
Print(n[2:7:2]) #[3,5,7]
Print(n[4:2])  #[5,7,9]
Print(n[3:7])  #[4,5,6,7]
Print(n[8:2:-2])#[9,7,5]
Print(n[4:100]) #[5,6,7,8,9,10]
```

List vs mutability:-

→ List objects are mutable so we can modify the contents after creating it.

→ eg:-

```
n=[10,20,30,40]
Print(n)  #[10,20,30,40]
n[1]=777
Print(n)  #[10,777,30,40]
```

Traversing the elements of list:-

→ we can traverse the list elements using a loop.

eg:

Date
Page

HW

eg:

```
n= [0,1,2,3,4,5,6,7,8,9,10]
i=0
while i<len(n):
    Print(n[i])
    i=i+1
```

For loop

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for i in range(0,len(n)):
    ~~Print(n[i])~~
    Print(
```

0,1,2,3,4,5,6,7,8,9,10 -

#WAP to print the even numbers in a list

```
n=[]
for i in range(0,10,2):
    Print[n[i]]
```

```
n=[0,1,2,3,4,5,6,7,8,9,10]
for i in range(1,len(n)):
    if(n[i]%2==0):
        Print(n[i])
```

if zero is in between elements
Condition
if(n[i]%==0 and n!=0)

Functions of list: —

1) len(): - returns the no. of elements in the list
eg:- n=[10,20,30,40]
Print(len(n)) ==> 4

2. count(): - It returns the number of occurences of a specified item in the list —
eg:- n=[1,2,2,2,2,3,3]
Print(n.count(1))
Print(n.count(2))
Print(n.count(3))
Print(n.count(4))

3. index():- This function returns the index of first occurrence of a specified element.

eg:-

```
n=[1,2,2,2,2,3,3]
Print(n.index(1)) => 0
Print(n.index(2)) => 1
Print(n.index(3)) => 5
Print(n.index(4)) => Value Error:
```

II Manipulating the list:-

→ The following functions are used to change the elements after creating the list:-

1. append():- append function is used to add item at the end of the list.

eg:-

```
list=[]
list.append("A")
list.append("B")
list.append("C")
Print(list)
# ['A','B','C']
```

2. insert():- This function is used to insert an element in the list at specific position.

eg:-

```
n=[1,2,3,4,5]
n.insert(1,888)
Print(n)
# [1,888,2,3,4,5]
```

→ eg:-

```
n=[1,2,3,4,5]
n.insert(10,777)
n.insert(-10,888)
Print(n) # [888,1,2,3,4,5,777]
```

* If the specified index is greater than the max index then the element inserted at last.
* If the specified index is smaller than the min index then the element inserted at last

3 extend()!- This function creates another list using all the elements of the first list.

eg!-

```
order1 = ["chicken", "Mutton", "Fish"]
order2 = ["RC", "KF", "FO"]
order1.extend(order2)
Print(order1)
["chicken", "Mutton", "Fish", "RC", "KF", "FO"]
```

eg!-

```
order = ["Chicken", "Mutton", "Fish"]
order.extend("Mushroom")
Print(order)
# ['Chicken', 'Mutton', 'Fish', 'M', 'U', 'S', 'H', 'R', 'O', 'O', 'M']
```

4. remove()!- It will remove an element from the list if it is present multiple times then 1st occurrence will be removed.

```
n = [10, 20, 10, 30]
n.remove(10)
Print(n)    # [20, 10, 30]
```

Note)
if the value provided to the remove function is not present then we will get the value error

5. POP()!- It also removes the elements from the list but only the last element at one time.
→ This function manipulates the list & returns an element.

eg!-

```
n = [10, 20, 30, 40]
Print(n.pop())  # 40
Print(n.pop(1))  # 30
Print(n)  # 10, 20
```

Note:-
If the list is empty then pop() function will return index error.

Note:
Out of the five functions in list the first 3 i.e. insert(), append(), extend() increases the list elements & the functions remove() & pop decreases the size of the list.

Ordering elements in list:-

1. reverse():- It will reverse the elements of the list

eg:- n=[10,20,30,40]
n.reverse()
Print(n) # [40,30,20,10]

2 sort():- It will sort the elements of the list
→ For numbers default sorting order is ascending & for strings the default sorting order is alphabetical & ascending.

eg:-
n=[20,5,15,10,0]
n.sort()
Print(n) # [0,5,10,15,20]

eg:-
n=[10,20,"A","B"]
n.sort()
print(n) # Type error.

Note:

To sort in reverse of default natural sorting order:-

→ we can sort the elements in reverse of 'default' natural sorting order by using reverse=True argument inside the sort() function

eg:-

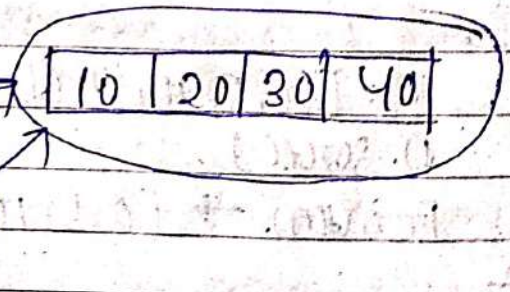
```
n=[40,10,30,20]
n.sort()
print(n) #[10,20,30,40]
n.sort(reverse=True)
print(n) #[40,30,20,10]
n.sort(reverse=False)
print(n) #[10,20,30,40]
```

Aliasing & cloning of list:-

→ we can clone or generate another reference of an existing list which is known as aliasing of list

eg:

```
x=[10,20,30,40]
y=x
print(id(x))
print(id(y))
```

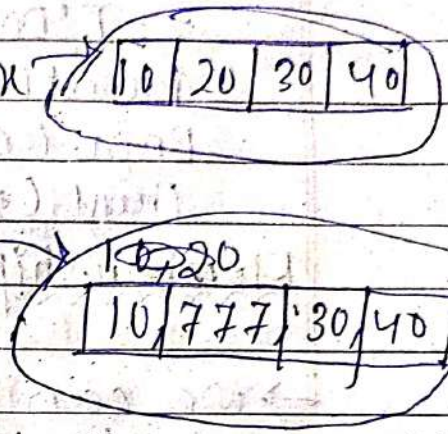


→ When we do aliasing two reference variable will get same reference object. When we change value of the object using one reference variable the other also changes without information

→ In order to overcome this problem we can use Cloning.
→ Cloning is the process of creating exactly duplicate independent object using slice operation or copy() function.

By using slice operators.

```
x = [10, 20, 30, 40]
y = x[:]  / y = x.copy()
y[1] = 777
Print(x) ⇒ [10, 20, 30, 40]
Print(y) ⇒ [10, 777, 30, 40]
```



Mathematical Operators :- 21-20-01-2022

The following operators can be used as to perform mathematical operations on list & they are

i) Concatenation (+)
ii) Repetition (*)

1. Concatenation Operator (+) :-

→ We can use + to concate 2 list.

eg:
```
a = [10, 20, 30]
b = [40, 50, 60]
c = a + b
Print(c) ⇒ [10, 20, 30, 40, 50, 60]
```

2. Repetition Operator (*) :-

→ We can use the * for repeatation of a list.

eg: x = [10, 20, 30]
```
y = x*3
Print(y) ⇒ [10, 20, 30, 10, 20, 30, 10, 20, 30]
```

Comparing list objects:-

→ We can use the relational operators as the comparision operator.

eg:-

```
x=["Dog","Cat","Rat"]
y=["Dog","cat","Rat"]
z=["DOG","CAT","RAT"]
Print(x==y) #True
Print(x==z) #false
Print(x!=z) #True
```

Membership Operator:-

→ We can check whether an element belongs to a list or not using the membership operator.

→ We can use in & not in operator for checking membership.

eg:

```
n=[10,20,30,40]
Print(10 in n)      #True
Print(10 not in n)  #false
Print(50 in n)      #false
Print(50 not in n)  #True
```

clear() function:-

→ We can use this function to delete all the elements in the list.

eg:

```
n=[10,20,30,40]
Print(n) #[10,20,30,40]
n.clear()
Print(n) #[]
```

Nested list:-

→ one list inside another list it's called nested list.

eg:- n = [10, 20, [30, 40]]

```
print(n)      # [10, 20, [30, 40]]
print(n[0])   # 10
print(n[2])   # [30, 40]
print(n[2][0]) # 30
print(n[2][1]) # 40
```

Nested list as Matrix:-

→ In python we can represents matrices in the form of nested list.

eg:

```
n = [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
print(n)
print("Elements by Row wise:")
for r in n:
    print(r)
print("Elements by matrix style:")
for i in range(len(n)):
    for j in range(len(n[i])):
        print(n[i][j], end=' ')
    print()
```

Output

```
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]
Elements by row wise:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
Elements by matrix style:
10 20 30
40 50 60
70 80 90
```

Date
Page

List Comprehensions:-

→ Lists can be created by using any other collection of objects like tuple, range, list, dict etc, based on some condition.

→ To do this the syntax is:

list = [expression for item in list if condition]

eg:

```
s=[x*x for x in range(0,11)]
print(s)
#[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

eg:

```
m=[x for x in s if x%2==0]
print(m)
#[0, 4, 16, 36, 64, 100]
```

* WAP to display unique vowels present in the given word.

Tuple Datastructure:-

→ Tuple is exactly same as list the only difference is it is immutable.

→ Tuple provides the following properties & they are, 1) Insertion order is preserved

2) Duplicates are not allowed.

3) Heterogeneous objects are allowed.

4) It supports +ve & -ve both indexes to extract elements.

5) We can represent tuple using round bracket called as parenthesis.

eg: → Immutable

t = 10, 20, 30, 40

Print(t) # (10, 20, 30, 40)

Note:

The default data structure for a collection of elements in python is tuple.

tuple() function:-

We can create a tuple from a list using this function.

eg:

list = [10, 20, 30]

t = tuple(list)

Print(t) # (10, 20, 30)

t = tuple(range(10, 20, 2))

Print(t) # (10, 12, 14, 16, 18)

Accessing elements of tuple:-
→ We can access the elements either using index or slice operator
eg:
1. By using index:-

```
t=(10,20,30,40,50,60)
Print(t[0]) #10
Print(t[-1]) # 60
Print(t[10]) # index error: tuple index out of range
```

2. By using slice operator:-

```
t=(10,20,30,40,50,60)
Print(t[2:5])  #(30,40,50)
Print(t[2:100]) #(30,40,50,60)
print(t[::2])  #(10,30,50)
```

Tuple vs immutability
→ Tuple objects are immutable so once declared cannot be modified
eg:

```
t=(10,20,30,40)
t[1]=70  #
```

Mathematical operator for tuple:-

1. Concatenation operator(+):

```
t1=(10,20,30)
t2=(40,50,60)
t3=t1+t2
Print(t3) #(10,20,30,40,50,60)
```

2. Repetition operator (*):-

t1=(10,20,30)
t2= t1*3
Print(t2) # (10,20,30,10,20,30,10,20,30)

functions of tuple:-

1. len():- no. of elements present in the tuple
eg:- t=(10,20,30,40)
Print(len(t)) # 4

2. count():- no. of occurences of given elements in the tuple
eg:-
~~t=(10,20,30,40~~ t=(10,20,10,10,20)
Print(t.count(10)) # 3.

3. index():- returns index of first occurence of the given element
→ If the specified element is not available then we will get ValueError.
eg:
t=(10,20,10,10,20)
Print(t.index(10)) # 0
Print(t.index(20)) # ValueError.

4. sorted():- To sort elements based on default natural sorting order
eg:-
t=(40,10,30,20)
t1=sorted(t)
Print(t1) # (10,20,30,40)
Print(t) # (40,10,30,20)

Date
Page

- we can sort according to reverse of default natural sorting order, as follows

```
t1=sorted(t, reverse=True)
Print(t1) # [40, 30, 20, 10]
```

Note!
The sorted function converts the tuple into list after sorting.

5. min() and max() function:-
→ It returns minimum & maximum value in the tuple

eg: t=(40, 10, 30, 20)
Print(min(t)) #10
Print(max(t)) # 40

Dt - 31-01-2023

6. cmp() :-
- This function is used to compare datastructure tuple with one another.
- If both the tuples are same then it returns 0, if the first tuple is less than second tuple then -1 is returns else +1 is returned

eg:-
t1=(40, 20, 30)
t2=(40, 50, 60)
t3=(40, 20, 30)
Print(cmp(t1, t2)) #-1
Print(cmp(t1, t3)) # 0
Print(cmp(t2, t3)) #+1

Note:
cmp() function is available in python 2 but for python 3 we have to compare the elements using equality operator.

Tuple packing and unpacking:-

→ Tuple packing is the process of grouping more than one variables on into one tuple element.

eg:-

```
a=10
b=20
c=30
d=40
t=a,b,c,d
Print(t) # (10,20,30,40)
```

→ Tuple unpacking is the process of assigning different variables with the elements of the tuple.

eg:- t=(10,20,30,40)

```
a,b,c,d=t
Print("a=",a,"b=",b,"c=",c,"d=",d)
# a=10 b=20 c=30 d=40
```

Note:-

The number of variables used for assigning should match with the number of elements inside the tuple otherwise we will get value error.

Tuple Comprehension:-

→ Tuple Comprehension is not supported by python.

eg:- t=(x**2 for x in range(1,6))

```
Print(type(t))
for x in t:
    Print(x)
```

Output:<class 'generator'>

1
4
9
16
25

Note:-1
As tuple objects are immutable tuple comprehensions are not supported in python.

Note 2
- When the user use comprehension operations in tuple then PVM creates a new kind of object called generator object in place of tuple, which is similar to list object

* Difference between list and tuple:

List	Tuple
- List is represented by elements separated by comma & enclosed within square brackets (Compulsory)	- Tuple represents list of elements separated by comma enclosed with in round brackets (Optional)
- List is mutable	- Tuple is immutable
- Duplicate elements are allowed	- Duplicates are not allowed
- A list cannot be used for creating the key value of a dictionary	- A tuple can be used for creating the key value of a dictionary.

Set Datastructure:-
- Set datastructure is a collection of elements separated by comma, enclosed with in curly braces.
- We can use set to represent a group of unique elements.
- The set provides the following features.
  * Duplicates are not allowed
  * Insertion order is not preserved.
  * Indexing & Slicing are not allowed

* Heterogenous elements are allowed.
* Set objects are mutable.

Creation of Set objects:-

i) By using direct initialization:-

```
S={10,20,30,40}
Print(S)
```

ii) By using Set() function:-

- we can use the set function to create a set out of any collection.

Syntax:

```
S=set(any sequence)
```

eg1:

```
l=[10,20,30,40,10,20,10]
S=set(l)
Print(S)
```

eg2

```
S=set(range(5))
Print(S) # {0,1,2,3,4}
```

Note:-

we cannot create a blank set because if we write S={} it is treated as a dictionary

Important functions of set:-

i) add(x):- This function will add an element to the set

eg: S={10,20,30}

```
#Print(S.add(40))
S.add(40)
Print(S)
```

ii) update(x,y,z):-

- This function add multiple items to the set.
- we can give elements or other collection objects as

Date
Page

ii) argument to the set

```
S={10,20,30}
l=[40,50,60,10]
S.update(l,range(5))
print(S)
```

iii) copy():

- This function creates a the copy of a set
- It is a cloned object

```
eg:- S={10,20,30}
     S1=S.copy()
     Print(S1)
```

iv) POP():-

- It will remove an element from the set randomly and returns a set

```
S={40,10,30,20}
Print(S)
Print(S.POP())
Print(S)
```

v) remove(x):-

- It removes specified element from the set
- If the specified element is not present then we will get key error.

```
eg: S={40,10,20,30}
    S.remove(30)
    Print(S) # {40,10,20}
    S.remove(50) # KeyError: 50
```

vi) discard(x):-

- It removes the specified element from the set
- If the specified element not present then the set then we wont get any error

S={10,20,30}
S.discard(10) ~~{20,30}~~
~~S.discard(50)~~
Print(S) ⇒{20,30}
S.discard(50)
Print(S) ⇒{20,30}

vii) clear():-
→ It removes all the elements from the set
eg. S={10,20,30}
Print(S) # {10,20,30}
S.clear
Print(S) # set()

Mathematical operations on the set:-

i) union():-
- This function will combine all the elements of both the sets & print it.
eg:-
x={10,20,30,40}
y={30,40,50,60}
Print(x.union(x,y)) #
Print(x|y)

ii) intersection():-
- It returns all the common elements in both the set
eg: x={10,20,30,40}
y={30,40,50,60}
Print(x.intersection(y))
Print(x&y)

iii) difference():-
- It will returns the elements of the first set but not present in the second set.
eg:- x={10, 20, 30, 40}
y={30, 40, 50, 60}
Print(x.difference(y)) #{10, 20}
Print(x-y) #{10, 20}
print(y-x) {50, 60}

iv) Symmetric_difference():-
- This will returns the elements present in set 1 & 2 but not in both.
eg:-
x={10, 20, 30, 40}
y={30, 40, 50, 60}
Print(x.Symmetric_difference(y)) #{10, 50, 20, 60}
Print(x^y) #{10, 50, 20, 60}

Membership Operators:- (in, not in)

- we can use the membership operator to check the presence of an element or to traverse with in the set.
eg1:- s=set("durga")
Print(s)
Print("d" in s) # True
Print("z" in s) # false
eg2:- s=set("durga")
for i in s:
Print(i) #{'d', 'r', 'u', 'a', 'g'}

Set comprehension:-

- Set comprehension is possible.

```
s={x*x for x in range(5)}
print(s) #{0,1,4,9,16}
s={2*x for x in range(2,16,2)}
Print(s) #{16,256,64}
```

Set objects won't supporting indexing and slicing:-

- Set objects don't provide indexing or slicing so inorder to access the elements of set we should use the membership operator.
- For accessing the elements we can write

```
fis=set("durga")
for i in s:
    print(i)
```

Dictionary Datastructure:-

- Dictionary datastructure is represented by a group of objects in form of key-value pairs enclosed with in curly braces.
- The dictionary is represented as

```
d={key:value,key:value}
eg:
d={100:'durga',200:'ravi',300:'shiva'}
```

- Dictionary supports the following features.
  - * Duplicate keys are not allowed but duplicate values are allowed
  - * Heterogeneous objects are allowed
  - * Insertion order is not preserved
  - * Dictionaries are mutable.
  - * Dictionaries are dynamic
  - * Indexing & slicing not supporting.

Creating dicionary:-

- We can create a dictionary by the following ways:

i) By direct initialization

eg

d = {100: 'durga', 101: 'ravi', 102: 'shiva'}

ii) By using dictionary function:-

dict()

- We can use a dictionary function to create a dictionary by providing two arguements to the function.
- first one can be a tuple & second one can be a tuple or list

eg

l = [10, 20, 30]

t = (50, 60, 70)

d = dict(t:l)

Print(d) {50: 10, 60: 20, 70: 30}

- for creating an empty dictionary we can write d={} or d=dict()

accessing the elements:-

- We can access the elements of the dictionary by using the key value.

eg: d={100: 'durga', 200: 'ravi', 300: 'shiva'}

Print(d[100]) # durga

Print(d[300]) # Shiva

Note: If the specified key is not present we will get key error.

has_key():-

- This function is used to check if the key is available in the dictionary or not.
- It will return 1 if it is present available otherwise returns zero.

eg: d.has_key(4w) # returns 0.

Updating dictionaries:-

- we can update a dictionary value through its key.
- If the key is not available then a new pair will be added to the dictionary.

eg:

```
d={100:"durga",2w:"ravi",3w:"shiva"}
Print(d) # {1w:"durga",2w:"ravi",3w:"shiva}
d[4w]="Pavan"
Print(d)#{1w:"durga",2w:"ravi",3w:"shiva",4w:"pavan"}
d[1w]="Sunny"
Print(d)#{1w:"Sunny",2w:"ravi",3w:"shiva",4w:"pavan"}
```

Dt-02-02-2023

Deleting elements from dictionary:-

→ we can delete any element from the dictionary by using delete function specified with key value.

→ This will delete the value associated with the key.

eg:-
d {1w:"deerga", 2w:"ravi", 3w:"shiva"}
Print(d) #{1w:"deerga", 2w:"ravi", 3w:"shiva"}
del d[1w]
Print(d): #{2w:"ravi", 3w:"shiva"}
del d[4w]

Note:
If the key provided is not available then this will return Key error.

clear():-
→ This function will clear all the elements from the dictionary.
→ eg:-
d={1w:"deerga", 2w:"ravi", 3w:"shiva"}
Print(d) #{1w:"deerga", 2w:"ravi", 3w:"shiva"}
d.clear()
Print(d): #{}

del d:-
→ This will completely delete the dictionary with the structure.
eg:-
d={1w:"deerga", 2w:"ravi", 3w:"shiva"}
Print(d) #{1w:"deerga", 2w:"ravi", 3w:"shiva"}
del d
Print(d) # NameError: name 'd' not defined

Important functions of dictionary:-

i) dict():-
This function is used to create a dictionary in multiple ways

d=dict() → # creates empty dictionary,
d=dict({1:"durga",2:"ravi"}) → # creates dictionary with specified element
~~d=dict([(1,"durga"),(2,"ravi"),(3,"shiva")])~~

d=dict([(1,"durga"),(2,"ravi"),(3,"shiva")])
→ # creates dictionary with the given list of tuple elements

ii) len():- returns the no. of elements in the dictionary.

iii) clear():- To remove all elements from the dictionary

iv) get():- This will return the value associated with a key
→ The user has to provide the key as argument and the function will return the value associated to it. If the provided key is not present then it will return none as result without giving any error.

eg:
d={1:"durga", 2:"ravi", 3:"shiva"}
Print(d[1]) # durga
Print(d[4]) # KeyError: 4
Print(d.get(1)) # durga
Print(d.get(4)) # None

(iv.) d.get(key, default value):
→ If the key is not available in the dictionary then instead of displaying none get function will print the default values.

eg:
print(d.get(101, "Guest")) ⇒ durga.
print(d.get(104, "Guest")) ⇒ Guest

iv) POP():-

→ This will remove the element specified as in the function.
→ The removal will be done according to the key provided
→ If the key is not present then you will get key error.

eg:
d={101:"durga", 201:"ravi", 301:"shiva"}
print(d.pop(101)) # durga
print(d) # {201:"ravi", 301:"shiva"}
print(d.pop(404)) # KeyError: 404

v) Popitem():
It removes an item from the dictionary & returns it.

eg: d={101:"durga", 201:"ravi", 301:"shiva"}
print(d) # {101:"durga", 201:"ravi", 301:"shiva"}
print(d.popitem()) # (301:"shiva")
print(d) # {101:"durga", 201:"ravi"}

vi) Keys():-
→ It returns all the keys values in the dictionary.

eg: d: {1:"durga", 2:"ravi", 3:"shiva"}
Print(d.keys()) # dict_keys([100, 200, 300])
for k in d.keys():
Print(k) # 100
200
300

VII) values():
It returns the values present in the dictionary
eg:
d: {1:"durga", 2:"ravi", 3:"shiva"}
Print(d.values()) # dict_values(['durga', 'ravi', 'shiva'])
for v in d.values():
print(v) # durga
ravi
shiva

VIII) items():-
It will returns list of tuples representing key-value pair.
eg: d: {1:"durga", 2:"ravi", 3:"shiva"}
for K,V in d.items():
Print(K,"--", V)
Output
1 -- durga
2 -- ravi
3 -- shiva

IX) copy():-
To create exactly duplicate dictionary (Cloned copy)
d1=d.copy();
X) setdefault():
This function returns the corresponding value if the specified key is present.
If the key is not available then it will create

a new item in the dictionary by using the key & value specified

eg:

```
d={1:"durga",2:"ravi",3:"shiva"}
print(d.setdefault(4,"pavan")) #Pavan
print(d) #{1:"durga",2:"ravi",3:"shiva",4:"Pavan"}
print(d.setdefault(1,"sachin")) #durga
print(d) #{1:"durga",2:"ravi",3:"shiva",4:"Pavan"}
```

xi) update():-

→ This function will update the values provided by the argument in another dictionary.

eg:

```
d.update(x)
```

All items present in the dictionary 'x' will be added to dictionary d.

Dictionary Comprehension:-

→ Comprehension concept is not available for dictionary.

→ If we are using comprehensions for dictionary the generator object will be used to support the operations.

eg:

```
squares={x:x*x for x in range(1,6)}
print(squares)
double={x:2*x for x in range(1,6)}
print(double)
#{1:1,2:4,3:9,4:16,5:25}
{1:2,2:4,3:6,4:8,5:10}
```

Function, Module & package:-

Functions:

→ Function is a set of instructions design to perform a particular task.
→ All the statements must be inside a single unit or block. This block or unit is called function.
→ The advantage of using function is code reusability
* code reusability represents writing the code once & executing it multiple times

Types of functions:-

→ Python supports two types of functions
i) Built in
ii) user-defined

i) Built-in functions:-
This represents all the functions provided by python software.
eg: clear()
id()
update() ... etc

ii) Userdefined functions:-
This functions will be created by the user to perform such tasks which are not provided by in built function.
→ In order to create a user defined function we will use a keyword "def" (mandatory).
→ The structure of user defined function will look like,

Syntax:

```
def function_name(Parameters):
    """doc string"""
    ==
    return value.
```

→ The parameters here will work as formal arguments. So we have to provide the values while calling the function through actual arguements.

Note:
In the function structure def keyword is mandatory, return & parameters are optional.

eg:- write a function to print hello.

```
def wish():
    Print("hello")
wish()
wish()
wish()
```

Parameters:-

→ Parameters are the values required by the function to operate.

→ If the function declaration contains parameters then we have to provide appropriate values while calling the function; otherwise we will get error

eg:

```
def SquareIt(number):
    Print("The square of ", number, "is", number*number)
SquareIt(4) # The square of 4 is 16
SquareIt(5) # The square of 5 is 25
```

Date
Page

Return Statement:-

→ If the return statement is present in a function then while calling the function it should be initialized because the function will return some result

eg

```
def add(x,y):
    return x+y
result = add(10,20)
Print("The sum is:", result)    # The sum is 30
Print("The sum is:", add(100,200)) # The sum is 300
```

Note

The default value for return statement is none.

Returning multiple values from a function:-

→ In python we can return multiple values using a single return statement.

→ In order to return multiple values write return and all the variables separated by comma.

eg:

```
def sum_sub(a,b):
    sum = a+b
    sub = a-b
    return sum, sub.
x,y = sum_sub(100,50)
Print("The sum is :", x) # The sum is: 150
Print("The substraction is:", y) # The substraction is: 50.
```

Date 3/02/2023
Page

Types of arguments:

→ Python provides two different types of argument that can be used with function that call & declaration.
→ The arguments that are used with function declaration are known as formal arguments & the arguments that are used in function call are known as actual arguments.

eg
```
def f1(a,b):
    - - -
    - - -

f1(10,20)
```

→ The actual arguments in python can be of 4 different types.
→ The different actual argument types are
1) Positional arguments
2) keyword arguments
3) Default arguments
4) variable length arguments

1) Positional arguments:-
→ These are the arguments passed to a function in positional order.
→ The position & value should match; otherwise the output may vary.

eg
```
def sub(a,b)
    Print(a-b) #-10 #10
sub(10,20)
sub(20,10)
```

Note.
we must provide the required number of arguments to the function call, otherwise we will get error.

2. Keyword arguments
→ In this we will pass the arguments by using keywords having the value.
→ The sequence of the arguments doesnot matter.
eg

```
def wish(name,msg):
    Print("Hello", name, msg)
wish(name="Durga", msg="Good Morning")
wish(msg="Good Morning", name="Durga")
```

Output
Hello Durga Good Morning.
Hello Durga Good Morning.

3. Default arguments :-
→ Using this we can pass default values for positional arguments
→ When the user doesnot provide the value for the argument then Default value will be used.
eg

```
def wish(name="Guest")
    Print("Hello", name, "Good Morning").
wish("Duga")
wish()
```

Hello Durga good Morning.
Hello guest good morning.

4. Variable length arguments
→ Sometimes we may want multiple values to be passed into a function to a single argument
→ In this case we will declare a variable length

that can take multiple values at a time.
→ The variable length argument can be declared as def f(*n):

Eg

```
def sum(a*n):
    total = 0
    for i in n:
        total = total + i
    Print("The sum is=", Total)
```

Sum()	# The sum = 0
Sum(10)	The sum = 10
Sum(10, 20)	The sum = 30
Sum(10, 20, 30, 40)	The sum is: 100

Types of Variables:-

→ Python provides two kinds of variable, bun operation
1) Global variable
2) Local variable

1) Global variable:-

→ The global variables can be defined outside of every function so that they can be accessed in all the functions.

→ eg:

```
a=0 # Global variable
def f1():
    Print(a)
def f2():
    Print(a)
f1()  # 10
f2()  # 10
```

2) Local variable.

- A variable defined inside a function is known as local variable.
→ The local variable will be accessed inside the function where it is declared.

⇒ eg

```
def f1():
    a=10
    print(a) # valid  # 10
def f2()
    print(a) # invalid # a is not defined
```

Global Keyword -

- This keyword will be used for two different purposes
→ To declare a global variable inside a function.
- To make global variable available to the function so we can perform modification.
→ we can use global keyword to define a global variable inside the function.

eg a=10

```
def f1():
    global a
    a=777
    print(a)
def f2():
    print(a)
f1()   #777
f2()   #777
```

Note:
When the name of global variable & local variable having same name then we can access global variable inside the function by writing

Eg
```
a = 10 # Global variable
def f1():
    a = 777 # local variable
    print(a) # 777
    print(globals()['a']) # 10
f1()
```

Recursive functions:-
- A function calling itself is known as recursive function.

Eg
~~factorial~~

Advantages:-
- It reduces the length of the code & improves readability
- Complex problems can be easily solved

Q WAP to find factorial of a number using recursion
```
def factorial(n):
    if n==0:
        result = 1
    else:
        result = n * factorial(n-1)
    return result
print("factorial of 4 is: ", factorial(4))
print("factorial of 5 is: ", factorial(5))
```

Factorial(n): n*Factorial(n-1)

01-07-02-2023

Anonymus Functions:-

→ The functions without any name is called anonymus function.
→ Sometimes we use lambda functions as anonymus function.
→ The purpose of anonymus function is constant and single use.
eg:-

```
S= lamda n: n*n
Print("The square of 4 is:",S(4))  # The square of 4 is: 16
Print("The square of 5 is:",S(5))  # The square of 5 is: 25
```

Note:-
The lambda function is basically used to define an instant function with the keyword lambda.

filter functions:-

→ Filter function is used to filter the values from a given collection according to specified condition.
→ The filter function can be used as: filter(function, sequence)
eg without lamda function:-

```
def isEven(x):
    if x%2 == 0:
        return True
    else:
        return False
l=[0,5,10,15,20,25,30]
l1= list(filter(isEven,l))
Print(l1)  # [0,10,20,30]
```

2. Using lambda function:

```
l=[0,5,10,15,20,25,30]
l1=list(filter(lambda x:x%2==0,l))
print(l1) # [0,10,20,30]
```

map() function:-

→ map() function is used to apply some functionalities to all the elements or some of the elements in a sequence and retrieve the new values after the modification.
→ The map() function will be used to modify the elements at a time in single instance.
→ The map() function can be used in following way:

Syntax

```
map(function, sequence)
```

#WAP to double all the list element present in the list

i) without lambda:-

```
l=[1,2,3,4,5]
def doubleit(x):
    return 2*x
l1=list(map(doubleit,l))
print(l1) #[2,4,6,8,10]
```

ii) with lambda:-

```
l=[1,2,3,4,5]
l1=list(map(lambda x:2*x,l))
print(l1) # [2,4,6,8,10]
```

* any():- It is a built in python function that returns true if at least one element of an iterable object is true else false.

Syntax

```
any(iterable)
```

reduce() function:-

→ This function is used to reduce a sequence of elements into a single element by applying specified function.
→ This function is present in functools module which needs to be imported before
→ The reduce() function can be used in the following way.

Syntax:

reduce(function, sequence)

eg

```
From functools import *
l=[10,20,30,40,50]
result = reduce(lambda x,y: x+y,l)
Print(result) # 150.
```

function Aliasing:-

→ function aliasing represents giving a new name to an existing function.
→ The execution of the result remains same, along with the memory allocated.

eg

```
def wish(name):
    Print("Good Morning:",name)
greeting = wish.
Print(id(greeting))
Print(id(wish))
Print(id(greeting))
greeting('Durga')
wish('Durga')
```

O/P:
```
#4429336
#4429336
# Good morning: Durga
# Good morning: Durga
```

Nested functions:-

→ When we will define one function inside another function then it is called nested function.
→ The inner function will execute first then outer function will execute.
→ Call the inner function inside the body of outer function.

eg

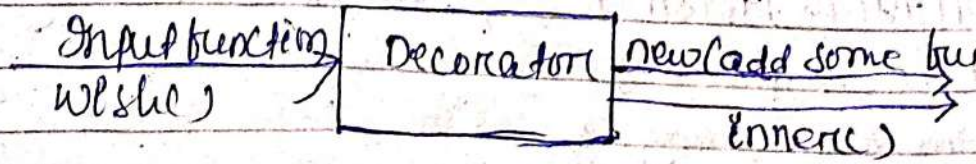
```
def outer():
    Print("outer function started")
    def inner():
        Print("inner function executed")
    Print("outer function is calling inner function")
    inner()
outer()
```

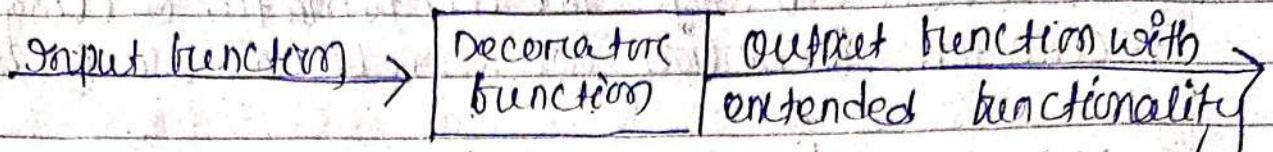
Output:-

outer function started
outer function calling inner function
inner function execution

function Decorators:-

→ These are used for a function that takes decorators as an argument and extends the functionality of decorator to the functions & returns the modified result.





→ The purpose of decorator function is without modifing the original function we can extends its functionality by providing some conditions in decorator function.

eg 1) def decor(func):
    def inner(name):
        if name == "sunny":
            Print("Hello sunny Bad Morning")
        else:
            func(name)
    return inner
@decor
def wish(name):
    Print("Hello", name, "Good Morning")
wish("Durga")
wish("Ravi")
wish("Sunny")

Output
Hello Durga Good Morning
Hello Ravi Good Morning
Hello Sunny Good Morning

Note
Whenever we call a normal function the decorator function will be automatically called.

2. In order to stop the automatic calling of decoration function we should not use @decor.

Calling a function with and without decorator:

→ decorfunction = decor(wish)

→ we can call a function manually by passing the function name as argument to decor func on whenever required.

eg:

```
def decor(func):
    def inner(name):
        if name == "Sunny":
            Print("Hello Sunny Bad Morning")
        else:
            func(name)
    return inner

def wish(name):
    Print("Hello", name, "Good Morning")
decorfunction = decor(wish) # Manual calling of decor function
wish("Durga")
wish("Sunny") # decoration won't be executed

decorfunction("Durga")
decorfunction("Sunky") # decor will be executed
```

output

Hello Durga Good Morning
Hello Sunky Good Morning
Hello Durga Good Morning
Hello Sunny Bad Morning.

Q WAP to maintain a bank statement where the user will enter the account number & initial balance. If the initial balance is >= 10,000 then account opening message will displayed otherwise account not opened message will be displayed.

```
def decorfunc():
    def acc(number):

def decor(func):
    def inner(acc no, balance):
        if balance >= 10000:
            Print("account opening") acc no, "account opened"
        else:
            func(acc no, balance)
    return inner.

def display(acc no, balance):
    Print("account opening")
decorfunction = decor(display)
display(12345678, 10000)
display(59876, 11,000)

    balance = int(input("Enter the balance:"))
    if (balance < 10000)
        display()
    else:
        decorfunction()
```

Decorator Chaining:-

→ When one decorator is used inside another decorator function then it is called as decorator chaining.
→ When we apply multiple decorator functions the inner functions will work first then the outer functions.
→ The decorator chaining can be done by writing the following.

Syntax:

```
@decor1
@decor
def num():
```

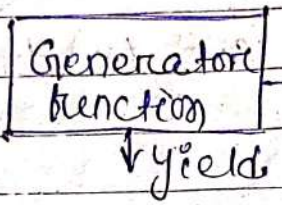
eg

```
def decor1(func):
    def inner():
        x = func()
        return x*x
    return inner
def decor(func):
    def inner():
        x = func()
        return 2*x
    return inner
@decor1
@decor
def num():
    return 10
print(num())
```

Date 9/05/2023
Page

Generators:-

→ Generator function is responsible for generating sequence of values.
→ Generator function is equivalent to ordinary function, the only difference is it uses a keyword → "yield" to return the values



eg def mygen():
    yield 'A'
    yield 'B'
    yield 'C'

g = mygen()
Print(type(g))

Print(next(g))
Print(next(g))
Print(next(g))
Print(next(g))

Output
<class 'generator'>
A
B
C

Note:
1) The line no 12 generated an error because it tries to access the 4th element which is not present in generator object.
2) next() is used to get the values inside a generator object

Q WAP to generate first n numbers where n will be entered by the user.

```
def firstn(num):
    n=1
    while n<=num:
        yield n
        n=n+1
values=firstn(5)
for x in values:
    print(x)
```

Output

1
2
3
4
5

Q WAP to generate fibonacci series upto n terms using generator function

```
def fibon(num):
    n1=0, n2=1
    print(n1)
    print(n2)
    for i in range(num):
        n3=n1+n2
        yield(n3)
        yield 'n3'
```

```
def fib():
    a,b=0,1
    while True:
        yield a
        a,b=b,a+b
for f in fib():
    if f>50:
        break
    print(f)
```

#: 0
1
2
3
5
8
13
21
34
55
89

Advantages of generator function:-

→ The generator function improves improve memory utilization of performance.
→ Using iterable objects becomes easy when generator function is used.
→ When reading data from large files generator function is useful.
→ Generator function is suitable for web scraping & ~~crawling~~ crawling.
→ over normal functions generator function supports comprehensions.

MODULES:-

- Modules are a collection of functions, variables & classes stored as a file.
→ Modules are used to provide some predefined activities to the user.
→ There are two different types of modules.
i) Builtin module.
ii) Userdefined module.

i) Built in modules:-
→ These are the modules already provided by the system i.e. python software.
→ The different built in modules are.
a Math, Calender etc.

ii) Userdefined modules:-
→ When the user defines a module on its own then it will be called as userdefined module.
→ In order to create a user defined module the user needs to write the complete code in a

Single file and save it using ".py" extension

eg: x=888
def add(a,b):
Print("Sum is:", a+b)
def product(a,b):
Print("Product is:", a*b)
→ save the file as Pratyush.py

Using a module:-
→ In order to access the contents of a module we can have two different ways
i) Using all the contents
ii) Using particular contents.

i) Using all the contents:-

→ In order to use all the contents we can use the import keyword like the following way.
Syntax:
import modulename.
→ After writing this line we can access the contents by writing module name. variable & modulename. function name.

eg import Pratyush
Print(Pratyush.x) # 888
Pratyush.add(10,20) # 30
Pratyush.product(10,20) # 200

ii) Using particular content:-

→ In order to use a particular content from a module we need to import only that content by writing

from modulename import the content

→ In order to access all the content we can use this from modulename import *

eg 1) From pratyush import x, add
Print(x) # 888
add(10,20) # 30
Product(10,20) # Error: name 'Product' is not defined

2) From pratyush import *
Print(x) # 888
Add(10,20) # 30
Product(10,20) # 200

Renaming a Module:
We can rename a module while importing it in the following way.

eg import pratyush as m

Various posibilities of import:-

import modulename
import module1, module2, module3.
import module1 as m
import module1 as m1, module2 as m2, module3 as m3.
From module import member
From module import member1, member2, member3.
from module import member1 as m
From module import *

Member aliasing:-

→ We can also create aothernames for the members in a module by using member aliasing.

→ once the aliasing is done you should use the new name for the members instead of the old name.

eg

1) From pratyush import a as y, add as sum
print(y) #888
~~Print~~(sum(10,20) # 30

eg2

from pratyush import a as y
Print(a) #NameError: name 'a' is not defined

Reloading a module:-

→ If the user imports the module multiple times by using multiple import statements then by default the module will be imported only once.

eg

~~import module1~~

module1.py:

Print("This is from module1")

test.py

import module1.
import module1
import module1
import module1
Print("This is test module");

Output

This is from module1
This is test module.

Note: In the above program module1 is imported only once instead of importing 4 times.

→ In order to reload a module we should explicitly load the module by writing reload function (reload()) for

→ The reload() function can be used in the following ways:

Syntax:

```
import imp
imp.reload(module1)
```

module1.py

```
Print("This is from module1")
```

test.py

```
import module1
import module1
from imp import reload
reload(module1)
reload(module2)
reload(module1)
Print("This is module")
```

Output

```
This is from module1
This is from module2
This is from module1
This is from module1
This is test module
```

Note

This reload function will be helpful when updating the content of a module in regular intervals

dir() function:→

→ This function is used to search or list out all the members present in a module.

→ This function will generate of list that contains name of the contents of a particular module.

→ dir() function can be used in two ways.

1) dir() → To list out all members of current module

dir(moduleName) → To list out all members of specified module

eg test.py

```
x=10
y=20
def f1():
    Print("Hello");
Print(dir()) # To Print all members of current module.
```

Output

['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'f1', 'x', 'y']

The special variable __name__:

→ For every python program some special variable name will added automatically.
→ The purpose of these variables are to store the execution information of the program.
→ We have a default variable called __main__ This will be added to a program if the program is executed as individual program.
→ If the program will execute as a module from some other program then the value of this variable will change according to the name of the module where it is defined.

module1.py

```
def f1():
    if __name__ == '__main__':
        Print("The code executed as a program")
    else:
        Print("The code executed as a module from some other program")
f1()
```

test.py
import module1
module1.f1()

Output
D:\Python_classes>py module1.py
The code executed as a program

D:\Python_classes>py test.py
The code executed as a module from some other program.
The code executed as a module from some other program.

DT/10/02/2022

math module:-

→ Python provides an in built module called math that provides different functions that can be used for operating mathematical operations.
→ The different functions provided by math module are

1. sqrt(x)
2. ceil(x)
3. floor(x)
4. fabs(x)
5. log(x)
6. sin(x)
7. tan(x)
...

Q WAP to implement the functions provided by the math module

1:
```
from math import *
Print(sqrt(4))      # 2.0
Print(ceil(10.1))   # 11
Print(floor(10.1))  # 10
Print(fabs(-10.6))  # 10.6
Print(fabs(10.6))   # 10.6
```

Note
To list out all the functions in math module we can use dir() function or help() function
eg import math.
help(math) / dir(math)

random module:-

→ This module is used to generate and operate with random values
→ This's module contains the following functions that can be used to generate random values.

1. random() function:-
→ This function always generates float values between 0 & 1
eg
```
from random import *
for i in range(10):
    Print(random())
```

2. randint() function:-
→ This function will generate random integer between two given numbers
→ Boundary-values are inclusive.
eg:
```
from random import *
for i in range(10):
    Print(randint(1, 100))
```

3. uniform():-
→ This function will return random float values between two given numbers.
eg
```
from random import *
for i in range(10):
    Print(uniform(1, 10))
```

4. randrange([start], stop, [step])
→ This function returns a range of numbers randomly in between the start & stop.
→ The start is inclusive but the stop value is exclusive.

st. start <= n < stop

→ Default value for start is 0 & default value for stop is 1

eg

randrange(10) ⇒ generates a number from 0 to 9

randrange(1,11) ⇒ generates a number from 1 to 10

randrange(1,11,2) ⇒ generates a number from 1, 3, 5, 7, 9

eg

```
from random import *
for i in range(10):
    Print(randrange(10))
```

5. Choice( )' function! –

→ This function will be used with list or tuple.

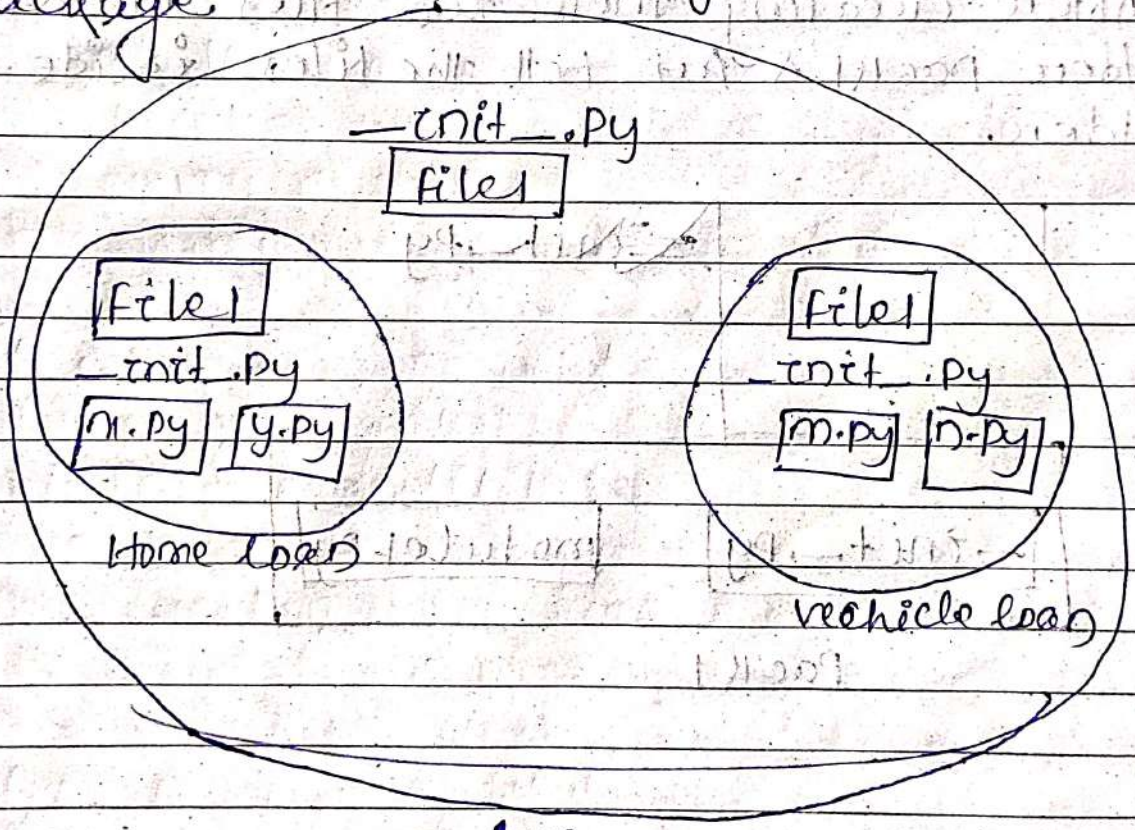
→ This will select a random element from the list or tuple & returns it

eg

```
from random import *
list = ["Sunny", "Bunny", "Chinny", "Vinny", "Pinny"]
for i in range(10):
    print(choice(list))
```

Packages:-

→ Packages are the result of encapsulation.
→ Package naming is the process of grouping different python modules of similar type into one folder.
→ If any folder in python contains a file "__init.py" then that folder will be treated as a package.
→ The __init__.py file can be empty.
→ Packages can contain sub packages also which is equivalent to one package inside another package.



Advantage of using Package:-
→ Naming conflict are resolved
→ We can identify the components uniquely.
→ Improves modularity of the application.

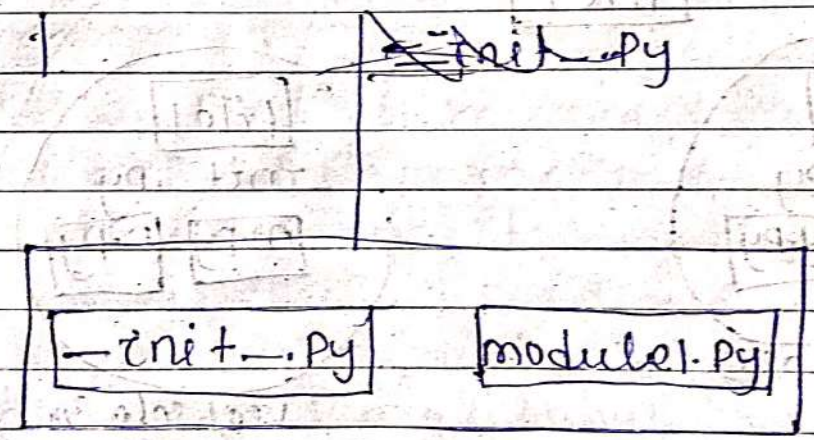
Creation of Package:-

→ We can create a package by writing following contents.

* __init__.py
  empty file

* module1.py
  def f1():
    Print("Hello this is from module1 present in Package1")

→ After creating these two files create a folder Pack1 & Put both the files inside the folder.



Accessing the contents of a package:-

→ We can access the package contents in two different ways

1) ~~test.py~~ 1st way:-

test.py

```
import pack1.module1
Pack1.module1.f1()
```

2) 2nd way:-

test.py

```
from Pack1.module1 import f1
f1()
```

Dt - 14-02-2023

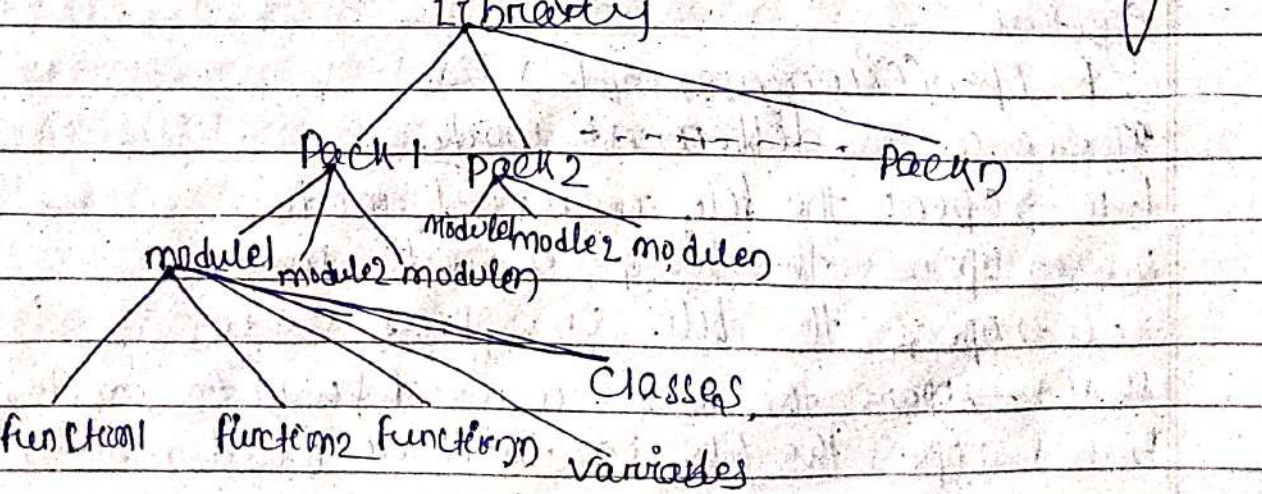
Defining & accessing Sub packages:-

→ A subpackage can be define in the same way of creating packages.

→ The sub package will be a package inside another package

→ The creation of modules inside the subpackage is equivalent to creating modules inside packages.

→ The definition & accessing is similar to packages.



UNIT-3

File Handling:-

→ In python we need file handling operations to store or retrive data permanently in a file.
→ Python provides two types of files to operate with i) Binary file
2) Text file.

1. Binary File:- The files that contains binary data like image, video, audio etc are treated as binary files.
2. Text files:- The files that contains text format data are known as text files.

Operation on a files:-

Opening a file:-

→ Before performing any operations we need to open a file. Ieeby using the inbuilt function provided i.e. open()
→ While opening the file we should specify the mode of the operation that represent the purpose of opening the file.
→ The different modes that can be provided to the open() function are
Syntax:-
f = open(Filename, mode)
There are different mode
1. r → opens the file in read mode
2. w → opens the files in write mode
3. a → opens the file in append mode
4. r+ → opens the file in read & write mode
5. w+ → opens the file in write or read mode.

6. a+ → opens the file in append & read mode
7. x → opens the file in exclusive creation mode for write operation.

Note:- All these files mode is also available for binary files. Only we have to combine it with opening mode for binary files.

Syntax eg. rb, wb, ab, r+b, w+b, a+b, x+b

f = open('abc.txt')

— The above eg. represents the textfile abc is opened for writing.

Closing a file:-

— After performing all the operation we can close the file by calling the close method

f.close();

Various properties of file object:-

— Once the file is opened we create a file object which has the following properties

→ The different properties a file object are

i) name :- represents name of opened file.

ii) mode :- represents the mode of opening the file

iii) closed :- returns a boolean value which indicates file is closed

iv) readable :- returns a boolean value whether the file is readable or not

v) writable :- returns a boolean value whether the file is writable or not

Eg

```
f= open("abc.txt",'w')
Print("File Name:", f.name)      # abc.txt
Print("File Name:", f.mode)      # w
Print("is File Readable:", f.readable())  # False
Print("is File writable:", f.writable())  # True
Print("is File closed:", f.closed)  # False
f.closed
Print("is File closed:", f.closed)  # True
```

Writing data to text files:-

→ After opening the file in any of the writing modes we can write data into the files by using two different methods

i) write method
ii) writelines method

Eg:-

```
f= open("abcd.txt",'w')          abcd.txt
f.write("Durga\n")               Durga
f.write("Software\n")            Software
f.write("Solutions\n")           Solutions
Print("Data written to the file successfully")
f.close()
# Data written to the file successfully.
```

Note:

The opening of the file in "w" mode, so every time the file will contains only the three lines provided in the program because the previous content will be overridden.

Reading characters from the text file:→
→ After opening the file in any of reading modes we can read the contents by using the following read methods.

read() → To read the total data from the file
read(n) → To read 'n' characters from the file
readline() → To read only one line
readlines() → To read all lines into a list

Eg: To read total data from a file:

```
f=open("abc.txt","r")
data=f.read()
print(data)
f.close()
```

O/P:- Durga
Software
Solution

Note:- read() & readlines() performs the same only difference is read() will provide the data according to the format of file but readlines() will provide the data in form of list

With statement:-
→ The use of with statement is it will create a block of operation statements for a file.
→ The with statement block automatically closes the file once the operations are over.
→ The user needn't close the file explicitly

Eg:
```
with open("abc.txt","w") as f:
    f.write("Durga\n")
    f.write("Software\n")
    f.write("Solution\n")
```

Print("Is file closed :", f.closed) #false
Print("Is file closed :", f.closed) #True

seek() and tell() methods:-

~~seek()~~:

tell():

→ This method returns the current position of the cursor from begining of the file.
→ The initial value of the cursor is zero.

eg f = open("abc.txt","r")
Print(f.tell())
Print(f.read(2))
Print(f.tell())
Print(f.read(3))
Print(f.tell())

abc.txt	output
Sunny	0
bunny	su
chinny	2
vinny	nny
	5

seek():-

→ The seek method will move the cursor from its position to the specified position.
→ The seek method can be used in the following way
f.seek(offset, fromwhere)
→ Offset - represent the no. of character used for seeking the cursor.
→ The fromwhere attribute can have three value 0 for the begining of file, 1 for the current position & 2 from the end of the file.

WAP to use the seek() method to seek the contents of a file.

→ data = "All Students are STUPID"
f = open("abc.txt", "w")
f.write(data)
with open("abc.txt", "r+") as f:
    text = f.read()
    Print(text)
    Print("The cursor position:", f.tell())
    f.seek(17)
    Print("The current cursor position:", f.tell())
    F.write("GENIUS!!!")
    F.seek(0)
    text = F.read()
    Print("Data After Modification:")
    Print(text)

Output:-
All Students are STUPID
The current cursor position: 24
The current cursor position: 17
Data After Modification
All Students are GENIUS!!!

#WAP to create a file with a string "Python programming is huge like python". change the word python in this string to java.
data = "Python programming is huge like python"
F = open("Pratyush.txt", "w")
f.write(data)
with open("Pratyush.txt", "r+") as f:
    text = f.read()
    Print(text)
    Print("The cursor position:", f.tell())
    F.seek(0)

```
f.write("Java!!!")
f.seek(32)
f.write("Java!!!")
f.seek(0)
text = f.read()
Print("Data After Modification")
Print(text)
```

checking the existence of a file:-

- we can check a file is particularly present on a system or not.
- for this we need "os library module" that provides a "sub module path" which contains a function "isfile()" to check whether the file is present or not
- The isfile() function can be used as
  os.path.isfile(filename)

Handling binary data:- DT-16-02-2023

→ we can use the same file operation functions changing the mode of opening files for handling binary files
⇒ Same open and close function will be used to open or close a binary file & write or read function will be used to write or read the data from a binary file
eg #Program to read image file and write it into a new file?

```
f1 = open("rosesum.jpg", "rb")
f2 = open("newpic.jpg", "wb")
bytes = f1.read()
f2.write(bytes)
Print("New image is available with the name: newpic.jpg")
```

Handling CSV files:-

→ When handling projects the user may need to handel the data in form of CSV files that represents comma Separated values

→ We can handel the CSV files by using a separate set of methods provided under CSV module.

→ CSV module provides read & writing both kinds of methods for writing & reading CSV files.

→ we can perform two operations on CSV files and they are: i) writing data to CSV files.
ii) Reading data from CSV file.

i) writing data to CSV file:-

```
import csv
with open("emp.csv", "w", newline="") as f:
    w=csv.writer(f) # returns csv writer object
    w.writerow(["ENO", "ENAME", "ESAL", "EADDR"])
    n=int(input("Enter number of employees:"))
    for i in range(n):
        eno=input("Enter Employee No:")
        ename=input("Enter Employee Name:")
        esal=input("Enter Employee Salary:")
        eaddr=input("Enter address:")
        w.writerow([eno, ename, esal, eaddr])
    print("Total employees data written to CSV File Successfully")
```

→ The CSV module provides two methods to write data into CSV files & they are 1) writer method.
2) writerow method

→ The writer method will create a CSV writer object that will write the data into CSV file.

- The writerow method will be used by the writer

Object to write data into csv file.

Note:
While opening the file if we specify newline attribute in the open function then blank lines will not be created between the data lines in excel sheet of csv file.

ii) Reading data from csv files:-

→ In order to read data we will need reader function that will read the data & returns it into a list.

Eg

```
import csv
f=open("emp.csv",'r')
r=csv.reader(f) # returns csv.reader object
data=list(r)
#Print(data)
for line in data:
    for word in line:
        print(word,"\t",end='')
    print()
```

Output

ENO	ENAME	ESAL	EADDR
1000	Durga	1000	Hyd
2000	Sachin	2000	Mumbai
3000	Dhoni	3000	Ranchi

Zipping & unzipping files:-

→ Zipping & unzipping file provide the following advantages

1. To improve memory utilization
2. We can reduce transport time
3. We can improve performance

→ In order to perform zipping & unzipping we will use a predefined module i.e. zipfile module which provides a class ZipFile

Creating zip files / zipping process:-
→ When ever we want to create a zip file we will call the file class obj. by providing three arguments.
1) file name.
2) mode of operation
3. ~~Zip deflated~~
3. ZIP_DEFLATED f= zipFile("file.zip", "r", "ZIP_~~STORED~~ DEFLATED")
4.

→ After creating the zip file class object we can add files into the zip folder by using write method.
→ eg: f.write(filename)
# WAP to create a zip folder that contains 3 files
from zipfile import *
f = zipfile("File.zip", "w", "ZIP_DEFLATED)
f.write("File1.txt")
f.write("File2.txt")
f.write("File3.txt")
f.close()
Print("~~f~~files.zip file created successfully")

Unzip Operation:-
→ We can create the file object that will unzip a zip file into the file named list.
→ In order to unzip we can create the zipfile object like f= zipfile("files.zip", "r", ZIP_STORED)
→ After creating the ZIP file object retrive the names of the files by writing eg:- names = f.namelist()

Eg

```
from zipfile import *
f=ZipFile("files.zip",'r',ZIP_STORED)
names=f.namelist()
for name in names:
    Print("File Name:", name)
    Print("The content of this file is:")
    f1=Open(name,'r')
    Print(f1.read())
    Print()
```

Python's Object Oriented Programming:-(OOPs)

- OOPs represents programs based on objects.
→ The control of the program can be done using a object in the program
→ The object orriendent programs represents two basic things and they are i) class
ii) object

i) Class:-

- class is a collection of object which collects the property & behaviour of the object
- class only contains the properties & behaviours of the object where property represent the variable & behaviour represents the methods or functions.

ii) Object:-

- Every class can be represented by a single entity called object.
- Object will be allowed to access the contents inside a class.
- When ever a content from the class is required object will be called for getting the information present inside the class.

Defining a class:

→ we can define a class by using def keyword & class keyword for defining the class & functions inside the class.
→ we can create a class in the following way

eg
```
class Student
    "This is Student class with required data"
    Print(Student.__doc__)
    help(Student)
```

Note
If we want to see, the contents of a class then we can get the details by writing
1. Print(classname.__doc__)
2. help(classname)

→ The contents we can provide in a class are variables & methods.
→ The variables in a class can be of 3 types
1) Instance variable
2) Static variable
3) Local variable.
→ The methods in a class are of 3 types
1) Instance methods
2) Class methods
3) Static methods

How to creat objects:-
→ We can create an object by using a reference variable.
→ Syntax:- referencevariable = classname()
Ex:- s = Student()
→ The reference variable will be used as an object to call the contents present in a class.
→ The reference variable has the access to the variables & methods present in a class.

Q WAP to create a student class and by creating object enter the student details & print it

```
class Student:
    def __init__(self, name, rollno, marks):
        self.name = name
        self.rollno = rollno
        self.marks = marks
```

```
def talk(self):
    Print("Hello My Name is:", self.name)
    Print("My Roll no is:", self.rollno)
    Print("My marks are:", self.marks)
S1=Student("Durga", 101, 80)
S1.talk()
```

Output

```
Hello My Name is: Durga
My Roll No is: 101
My marks are: 80
```

self variable:-

→ This variable represents a temporary reference object that points to the current object.
→ It works like "this" keyword in Java.
→ It can access the instance variable & instance methods of a class.

Constructor:-

→ In python we can provide constructor as a special method
→ The constructor in python has a name __init__(self)
→ The purpose of constructor is to initialize instance variables.
→ The constructor will execute automatically when one object is created.
→ If the user is not providing a constructor in a class, then python will automatically add a default constructor into the program.

eg

```
def __init__(self, name, rollno, marks):
    self.name=name
    self.rollno=rollno
    self.marks=marks
```

* WAP to show constructor & method execution in a class.

```
class Test:
    def __init__(self):
        Print("Constructor execution ---")
    def m1(self):
        Print("Method execution ---")
t1 = Test()
t2 = Test()
t3 = Test()
t1.m1()
```

Output :- Constructor execution ---
" "
" "
Method "

Note
In the above program 3 objects are created which calls the constructor 3 times. one obj. calls the method. So method m1() executes 1 time.

Types of Constructor :-
→ According to the no. of arguments the constructors are divided into 3 different types
i) default constructor
ii) Zero argument constructor
iii) Argumented constructor
→ The default constructor will be provided implicitly when the user doesnot provide a constructor but still creates an object using class.
→ The zero argument constructor is provided by the user generally to provide default or initial values for instance variable
→ The Argumented constructor will be used to provide the values in the object-creation statement

Date
Page

Eg:- Zero argument :- Argumented constr.

```
class Student                    class Student
    def __init__(self):              def __init__(self,a,b)
        a=0                              self.a = a
        b=0                              self.b = b
    t1=Student()                     t1 = Student(5,6)
```

Difference bet^n methods & constructor :-

Method	Constructor
- Name of method can be any name	- Constructor name should be always __init__
- It will execute when the user call the method	- It will execute automatically when the object is created
- For one object, one method can execute multiple time.	- For one object, constructor will executed once.
- It can contain any business logic.	- It is used to initialize instance variables.

Types of Variables :-

- We have three different types of variables according to their level of presence.

1. Instance variables (Object level variables)
2. Static variables (Class level variable)
3. Local variables (Block level variable)

1) Instance variables :-

- The variables that have directly related to object are instance variables.
- For every object we will get a separate copy of instance variable
- We can define an instance variable at three different locations.

i) Inside constructor by using self variables:-
→ when ever we write self.variablename inside a constructor that will be treated as an instance variable which will be automatically added to an object when created
eg def __init__(self):
    self.no = 100
    self.ename = 'durga'
    self.esal = 10000.

ii) Inside instance method by using self variables:-
→ If we write self.variable name inside any instance method then that variable will also be treated as an instance variable.
eg def m1(self):
    self.c = 30.

iii) Outside the class by using object reference variable:-
We can also define an instance variable outside the class after the object creation using object reference.variable name
eg t = Test()
   t.d = 40;

Accessing instance variables:-
We can access the instance variables using self variable inside the class & object reference variable outside the class.
eg class Test:
    def __init__(self):
        self.a = 10
        self.b = 20
    def display(self):
        print(self.a)
        print(self.b)
t = Test()
t.display()
print(t.a, t.b)

Output
10
20
10 20

Deleting the instance variable:-

- we can delete an instance variable by using del keyword.
→ for deleting the variable inside a class we will use self variable by first deleting it outside the class we will use object reference variable.

eg

```
class Test:
    def init_cnt(self):
        self.a=10
        self.b=20
        self.c=30
        self.d=40
    def m1(self):
        del self.d
t=Test()
Print(t.__dict__)
t.m1()
Print(t.__dict__)
del t.c
Print(t.__dict__)
```

Output

{'a':10, 'b':20, 'c':30, 'd':40}
{'a':10, 'b':20, 'c':30}
{'a':10, 'b':20}

Note

we can print the instance variable along with their values in dictionary format by using dict variable with the object name.

eg obj.name.__dict__.

Static var

2) Static variable! -

→ Static variables are related to class so they do not change from object to object.

→ Static variables are defined directly with the class outside of every method.

→ Static variable can be created once, & shared by all objects.

→ we can access the static variable by using the class name directly.

eg:

class Test:

s=10;

Test.s = 888

Difference between instance & static variable! -

- The instance & static variable are defined for objects & class.

- Instance variable will be allocated multiple times for multiple objects, but static variable will be allocated only once for multiple objects

Declaring static variable! -

- we can declare the static variables at 5 different positions & they are

1) within the class directly but from outside of any method

2) Inside the constructor by using class name.

3) Inside instance method by using class name.

4) Inside classmethod by using either class name or class variable.

5) Inside static method by using classname.

Eg class Test:
1) a=10
def __init__(self):
2) Test.b=20
3) def m1(self):
Test.c=30
@classmethod
4) def m2(cls):
cls.d1=40
Test.d2=400
@staticmethod
5) def m3():
Test.e=50
Print(Test.__dict__)
t=Test()
Print(Test.__dict__)
t.m1()
Print(Test.__dict__)
Test.m2()
Print(Test.__dict__)
Test.m3()
Print(Test.__dict__)
Test.f=60
Print(Test.__dict__)

Accessing the static variables:-
-We can access the static variables in the following 5 ways:-
1) inside constructor by using either self or classname
2) inside instance method by using either self or classname
3) inside class method by using either cls variables or classname
4) Inside static method by using classname
5) From outside of class by using either obj. reference or classname

deleting static variable:-
→ If we want to delete the static variable inside the class or outside the class & simply write
→ del classname.static variablename

3. Local variables:-
→ Local variables are limited to the blocks where they are created.
→ It will be created for that block & the block executes & will be deleted when the block completes its execution.
→ They can be accessed within that block only.

Setter and Getter Methods:-

Setter method
→ This method is used to set the values for the instance variables
→ we can use the setter method by writing.
Syntax def setVariable(self, variable):
self.variable = variable

Getter Method:-
- This method is used to get the values of Instance Variables.
→ we can use the getter method by writing:
Syntax:
def getVariable(self):
return self.variable

Note:-
When in a program constructor & setter both methods are present then the value provided by the constructor then the values will be overwritten overridden & new values will be apply.

Inner class:-

→ When we redefine one class inside another class then these types of classes are called innerclass

- When we want to interrelate object then we will use inner classes.
- The inner class object will only exist when the outer class object is create, so for creating & accessing the contents of inner class we need the outer class object.

eg

```
class Outer:
    def __init__(self):
        Print("Outer class object creation")
    class Inner:
        def __init__(self):
            Print("inner class object creation")
        def m1(self):
            Print("inner class method")
o = Outer()
i = o.Inner()
i.m1()
```

output
Outer class object creation
Inner class object creation
inner class method.

01-02-02-2023

Garbage Collection:-

→ Like other programming languages the object memory allocation needs to be deleted once the purpose of object is completed.

→ In other programming languages we have to manually delete the memory allocated for the objects.

→ In Python automatic deletion of object memory is done by garbage collector object

→we have to enable or disable the garbage collection process by writing the following statements:
1. To enable garbage collection:-
i) gc.isenabled():
→Returns True if GC enabled.
ii) gc.enabled()
→To enable GC explicitly
2. To disable the garbage collection
i) gc.disable():- To disable GC explicitly.

example
```
import gc
Print(gc.isenabled()) # True
gc.disable()
Print(gc.isenabled()) # False
gc.enable()
Print(gc.isenabled()) # True
```

Destructors:-
→Destructors are the special method that will destroy the object.
→The destructor name is written as __del__
→The destructor will be called by the garbage collector, to destroy the object & the garbage collector will perform the memory clean up process.
→~~The destructor~~
→When the user does not provide destructor in a program then garbage collector will automatically connects to a default destructor to destroy the object.

—The destructor can also be provided explicitly by the user in the program.

Eg

```
import time
class Test:
    def __init__(self):
        Print("Object Initialization...")
    def __del__(self):
        Print("Fulfilling last wish and performing clean up activities...")
t1=Test()
t1=None
time.sleep(5)
Print("End of application")
```

output

Object Initialization...
Fulfilling last wish and performing clean up activities...
End of application

Note

First the constructor of program will execute followed by the remaining contents & at the end destructor will execute

How to find number of references of object:-

→ We can count the total no. of references created for an object when we are creating object aliasing

→ We can count the total no of reference by using the method: sys.getrefcount(objectreferences)

```
import sys
class Test:
    Pass
t1=Test()
t2=t1
t3=t1
t4=t1
Print(sys.getrefcount(t1))
```

Output:- 5

Note :

for every object, Python internally maintains one default reference variable self.

Date
Page

Polymorphism:-

→ Polymorphism represent one element existing in multiple forms performing different every time.
→ The polymorphism can be represented by operators, functions, constructors etc.
→ We can see polymorphism in two different things & they are:-
1) Duck Typing
2) Method Overloading
3) Method Overridding
4) Construction overloading & overridding.

1) Duck Typing:-

→ In python we cannot specify the type explicitly.
→ In order to specify the types explicitly we should go for dynamically typed programs that supports duck typing.

```
→ def fl(obj):
      obj.talk()
```

2) Method overloading:-

→ The method overloading represent we can use the same method for different purposes.
→ The method overloading can be done by writing same method name by changing different arguments.
→ In python method overloading cannot be performing because if we define multiple methods with same name then python will consider the last method only.
→ The method overloading can be done in the following way:

```
eg class Test:
      def m1(self):
          Print("no-arg method")
      def m1(self,a)
          Print("one-arg method")
```

```
    def m1(self,a,b):
        Print("two-arg method")
t=Test()
#t.m1()
#t.m1(10)                    Output
t.m1(10,20)                  'two-arg method
```

Note:-

When we are executing the programs containing method overloading always call the last method present in the class.

How we can handle overloaded method requirements in Python:-

- we can perform the method overloading by taking the help of default argument or with variable no. of arguments.

1. Using default arguments:-

- we can perform method overloading by using default arguments in the method.
- To do this we can write:

```
class Test:
    def sum(self,a=None,b=None,c=None):
        if a!=None and b!=None and c!=None:
            Print("The sum of 3 Numbers:",a+b+c)
        elif a!=None and b!=None:
            Print("The sum of 2 Numbers:",a+b)
        else:
            Print("Please provide 2 or 3 arguments")
t=Test()
t.sum(10,20)                 ' Output
t.sum(10,20,30)              # The sum of 2 Numbers: 30
t.sum(10)                    The sum of 3 Numbers: 60
                             Please provide 2 or 3 arguments
```

2. Using variable no. of arguments:-
-We can also perform method overloading by using variable no. of arguments.

eg

```
class Test:
    def sum(self, *a):
        total=0
        for x in a:
            total = total + x
        Print("The sum:", total)
t=Test()
t.sum(10,20)          # The sum: 30
t.sum(10,20,30)       The sum: 60
t.sum(10)             The sum: 10
t.sum()               The sum: 0
```

Output

3. Method Overridding:-

→ When the parent class and child class contains the same method name with the same no. of arguments then method overridding will be perform by the child class object.

→ The child class object is always give preference to the child class method overridding the parent class method. This concept is called method overridding.

eg

```
class P:
    def property(self):
        print("Gold + Land + cash + Power")
    def marry(self):
        print("Appalamma")
class C(P):
```

```
    def marry(self):
        print("Pratyush")
c=C()
c.property()
c.marry()
```

Output-
Gold + Land + Cash + Power
Pratyush.

Note!
In the above example the child class contains two marry methods. Out of that the child class method overrides representing method overridding. If we want to execute the parent class overridden method then we can do it by writting the following statement in the child class method.

```
super().marry()  # Appalamma
```

4. Constructor overloading and ridding! —
Constructor overloading & ridding can be done by the same concept of method overloading & ridding

Constructor Overloading! —

```
Class Test!
    def __init__(self):
        print('No-arg Constructor')
    def __init__(self,a):
        print(" one-arg Constructor")
    def __init__(self,a,b):
        print("Two-arg Constructor")
#t1=Test()
#t1=Test(10)
t1=Test(10,20)
```

Output:- Two-arg constructor.

constructor overloading with default arguments:

```
class Test:
    def __init__(self, a=None, b=None, c=None):
        Print("constructor with 0|1|2|3 no. of arguments")
t1=Test()          # constructor with 0|1|2|3 no. of arguments
t2=Test(10)        "   "   "   "   "   "   "   "   "   "
t3=Test(10,20)     "   "   "   "   "   "   "   "   "   "
t3=Test(10,20,30)  "   "   "   "   "   "   "   "   "   "
```

ii) variable no of arguments:

```
class Test:
    def __init__(self,*a):
        Print("constructor with variable no. of arg")
t1=Test()
t2=Test(10)
t3=Test(10,20)
t4=Test(10,20,30)
t5=Test(10,20,30,40,50,60)
```

output

```
constructor with variable no. of arg
"   "   "   "   "   "   "
"   "   "   "   "   "   "
"   "   "   "   "   "   "
"   "   "   "   "   "   "
```

constructor overriding:-

```
class P:
    def __init__(self):
        Print("Parent constructor")
class C(P):
    def __init__(self):
        print("child constructor")
c=C()
```

output:- child constructor

Note:-
In the above example, if child class does not contain constructor then parent class constructor will be executed

Dt-03/03/2023

→ The overridden constructor of the parent class can be executed by using super() method in child class constructor.

eg

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
class Employee(Person):
    def __init__(self, name, age, eno, esal):
        super().__init__(name, age)
        self.eno = eno
        self.esal = esal
    def display(self):
        Print("Employee Name:", self.name)
        Print("    Age:", self.age)
        Print("    Number:", self.eno)
        Print("    Salary:", self.esal)
e1 = Employee('Durga', 48, 87245, 26000)
e1.display()
e2 = Employee('Sunny', 39, 87246, 30000)
e2.display()
```

Output

Employee Name: Durga	Employee Name: Sunny
Age: 48	Age: 39
Number: 87245	Number: 87246
Salary: 26000	Salary: 30000

Inheritance:-

→ Inheriting the properties & behaviours of parent class into child class is represented by inheritance.
→ The properties represent all the variables & the behaviour represents all the functions present in the Parent class which will move into the child class.
→ Inheritance can be done in the following way b/w the parent and child classes.

Syntax

```
Class Parent:
    ==
    ==
class child(Parent):
    ==      Inheritance.
    ==
```

Ex

```
class Student:
    a=10
    def m1():
        Print(a)
class Student1(Student):
    b=20
    def m2():
        Print(b)
        Print(super().a)
S1=Student1()
S1.m2()
S2=Student()
S2.m1()
```

O/P: 20
10
10

Note:

In the previous example class Student1 inherites the Properties of class Student, in order to access the content of parent class we can either create an object or access it using super() method.

Types of inheritance:-

→ We have the following kinds of inheritance based on the number and type of classes involved.

→ The different inheritances are i) single inheritance

i) Single inheritance (1 Parent class, 1 child class)

ii) multiple inheritance (multiple parent classes one child class)

iii) Heirarchical inheritance (one parent & multiple child classes)

iv) Multilevel inheritance (one Parent to one child to another child)

v) Hybrid inheritance (combination of any of the 4 inheritances)

Super() Method Keyword:-

→ super method will be used to access the content of parent class in child class before creating the object

→ super method will be used to access the following contents of parent class in child class.

1) Accessing parent class variable:-
2) Accessing Parent class Constructor.
3) Accessing Parent class Method.

1) Accessing Parent class variable:-

→ Before creating the object of child class if we want to access the parent class variable inside the body of child class then super() method is used

→ The super method will retrieve the values of the Parent class variable & provide it to the child class

→ This is also helpful when the parent class & child class contains the same variable name.

2) Accessing the parent class constructor:-
→ we can also access the parent class constructor inside the body of the child class constructor by using super() method.
→ The constructor accessing requires the no. of values for parent class as well as child class constructor. So while creating the object provide all the values sequentially.
→ Accessing the parent class constructor through super() method in child class constructor must be the 1st statement.

3). Accessing the Parent class method:-
→ we can access parent class method in child class by using super() method.
→ while accessing the parent class method it should be the 1st statement inside child class method.
→ If the method contains arguments then pass appropriate no. of values.

example

```
class Student:
    def init(self, a, b):
        self.a = a
        self.b = b
    def printing(self):
        Print(self.a)
        Print(self.b)
class Student1(Student):
    def init(self, a, b, c):
        super().init(a, b)  # Calling Parent class constructor
        self.c = c
    def display(self):
        super().printing()  # calling Parent class method
        Print(self.c)
Print(super().a)  # calling Parent class variable
S = Student1(10, 20, 30)
S.display()
S1 = Student1(50, 60, 70)
S1.display()
```

O/P

10 S 50 S1
20 60
30 70
10 20

About - DB
Advantage / des advantage

Date 9/08/2023
Page

Python Database Programming

→ In order to connect python programs to different databases we need some special modules & classes
→ In database programming we need the following classes & modules to connect different databases like Oracle, Mysql, Sqlserver, Gadfly, Sqlite etc.
→ For each database required a separated set of modules to connect programs to DB like
eg: i) cx_oracle module for communicating with oracle db.
ii) pymssql module for communicating with microsoft Sql Server

Connecting Python Programs to Oracle:-

→ In order to connect python programs to oracle database we need to follow the following steps.

Step1: import database specific module.
ex: import cx_Oracle.
Step2: Establish the connection by creating connection object by using connect() function.
ex: con = cx_Oracle.connect(database information)
i) The database information represents the username, password & the hostname.
ex: con = cx_Oracle.connect('scott/tiger@localhost')
Step3: To execute the sql commands & to hold the results some special type of objects required known as cursor object which can be created by using cursor() method.
ex: cursor = con.cursor().
Step4: Now the cursor object can take sql queries by using the following three methods.
i) execute(sqlquery) → To execute a single sql query.
ii) executescript(sqlqueries) → To execute a string of sql queries separated by semi-colon;
iii) executemany() → To execute a parameterized query.

eg cursor.execute("select * from employees")

Oracle DB

Step 5: After updating the sql queeries we have to comit or rollback the changes according to our requirement by using the following methods.

commit() → save the changes to the database.

rollback() → roll all temporary changes back.

Step 6: After the comit or rollback we can fetch the results from the cursor object using the following methods.

i) fetchone() → To fetch only one row

ii) fetchall() → To fetch all rows and it returns a list of rows

iii) fetchmany(n) → To fetch list n rows.

steps: eg 1: data = cursor.fetchone()
Print(data)

2: data = cursor.fetchall()
for row in data:
Print(row)

3: data = fetchmany(10)
Print(data)

Step 7: After all the operations close the connect created bw the database by using close method.

cursor.close()
con.close()

Working with Oracle database:-

→ We can work with oracle database by using the 7 different steps provided bw database programming

i) To connect with oracle data base:-

```
import cx_oracle
con = cx_oracle.connect('scott/tiger@localhost')
Print(con.version)
con.close()
```

output

11.2.0.2.0

2. WAP to creat an employe table in oracle database

```
import cx_Oracle
try:
    con = cx_Oracle.connect('scott/tiger@localhost')
    cursor = con.cursor()
    cursor.execute("Create table employees
                    (eno number, ename varchar2(10),
                    esal number(10,2), eaddr varchar2(10)
                    )")
    Print("Table created successfully")
except cx_oracle.DatabaseError as e:
    if con:
        con.rollback()
        Print("There is a problem with sql", e)
Finally:
    if cursor:
        cursor.close()
    if con:
        con.close()
```

Working with Mysql database:-

→ In order to work with mysql database we can have bocer default databases available in mysql & they are 1) information_schema
2) mysql.
3) Performance_schema.
4) test

Step1:- → we can use the mysql database by using the driver or connector information required for the python program, which can be done by writing import mysql.connector

Step2: After importing the connector we have to create the connection object by writing
con = mysql.connector.connect(database information)

Python Regular Expressions

→ A regular expression is a statement containing sequences of character that define a search pattern
→ eg: ^a...s$
→ The above statement defines a RegEx expression which finds a pattern in the string containing any five letter word that starts with a & ends with s.
→ Regular expressions are used to work with the programs faster
→ In order to work with regular expressions we should import a module called "re" which represents regular expression
→ This module provides different methods on sequence of characters that will be used to perform regular expression operations

Match() function:-

→ The re module provides a function called match that will use the pattern to match into the given string & according to the results the output will be given
→ The match() function returns a match object if a search is successful or the search is successful otherwise it returns 'None'.

1) WAP to use a regular expression for matching a string

→
```
import re
Pattern = '^a...s$'
test_string = 'abhas'
result = re.match(Pattern, test_string)
if result:
    Print("Search successful")
else:
    Print("Search unsuccessful")
```
O/P:
Search successful.

Note
The match() function takes two argument for work 1. The pattern
2. The test string to search.

Meta characters in expressions:-
→ Meta characters are the characters that are executed in a special way by RegEx engine.
→ The different meta characters are
[], ., ^, $, *, +, ?, {}, (), \, |

1. '[]' - Square brackets:-
→ These brackets are used to specify a set of characters provided inside the bracket.
→ It will consider each character provided inside the bracket individually

eg

Expression	String	Matched?
[abc]	a	1 match
	ac	2 match
	Hey Jude	No match
	abcdeca	5 Match

Note:- We can also provide a sequence of characters by using range inside square brackets
- [a-e] is the same as [abcde]
- [1-4] is the same as [1234]

→ we can also use the caret ^ symbol at the start of the square bracket to invert the searching.
- [^abc] means any character except a or b or c
- [^0-9] means any non-digit character.
~~→ we can also provide a period or m~~

2. '.' - Period:-
→ we can also provide a period or match as expression to the pattern.
→ each period is represented by a dot (.)

Expression	String	Matched?
.	a	No match
..	ae	1 match
...	aed	2 match
	acde	2 matches (containing 4 character)

3. '^' - Caret.
→ This symbol is used to check if the string starts with a particular character or not.

Expression	String	Matched?
^a	a	1 match
	abc	1 match
	bae	No match
^ab	abc	1 match
	acb	No match (starts with a but not followed by b)

4. '$' - Dollar
→ This symbol is used to check if a string ends with a particular character or not

eg

Expression	String	matched?
a$	a	1 match
	formula	1 match
	cab	No match

Dt: 10/03/2023

5 '*' - Star' :-
This character will be used to check 0 or more occurances of the pattern left to it

eg

Expression	String	Matched?
ma*n	mn	1 match
	man	1 match
	maan	1 match
	main	No match (a is not followed by n)
	woman	1 match

6 '+' - Plus' :-
This character will be used to check one or more occurances of the pattern left to it.

Expression	String	Matched?
ma+n	mn	No matched (no character)
	man	1 match
	maan	1 match
	main	No match (a is not followed by n)
	woman	1 match

7 '?' - Question Mark' :-
This character will match zero or one occurence of the pattern left to it.

Expression	String	Matched
ma?n	mn	1 match
	man	1 match
	maan	No match (more than one a characters)
	main	No match (a is not followed by n)
	woman	1 match

8 '{}' - Braces :-

→ This character will take two arguments : {n,m}

→ This code explains that at least n and at most m repetition of the pattern present left to it

Expression	String	Matched?
a{2,3}	abc dat	No match
	abc daat	1 match (at daat)
	aabc daaat	2 matches (at aabc and daat)
	aabc daaaat	2 matches (at aabc and daaaat)

→ We can also provide a range as a pattern to '{}' meta character

ex

Expression	String	Matched?
[0-9]{2,4}	ab123cde	1 match (match at ab123cde)
	12 and 345673	2 match (at 12 and 3456 73)
	1 and 2	No match

9. '|' - Alternation :-

→ This character is used for alternation.

eg

Expression	String	Matched?
a|b	cde	No match
	ade	1 match (match ade)
	acdbea	3 match (at acdbea)

→ The alternation operator matches the string that contains either a or b in alternation

10 '()' - Group:
- This meta character is used to group sub-Patterns
- eg (a|b|c)xz this pattern will match strings where a or b or c is present followed by xz
eg

Expression	string	Matched?
	abxz	No match
(a|b|c)xz	abxz	1 match (match at abxz)
	axzbcabxz	2 match (at axzbcabxz)

Special Sequence:-

* \A - Matches the string if the specified character is present at the start of the string.

Expression	String	Matched?
\Athe	the sun	Match
	In the sun	No Match

* \b - It will match if the specified character is present at the begining or ending of a word in the string

eg

Expression	String	Matched?
\bfoo	football	Match
	a football	Match
	afootball	No match
foo\b	the foo	Match
foo\b	foot	No match

* \B - It is working opposite to \b'
eg

Expression	String	Matched?
\BFoo	football	No match
	a football	No match
	afootball	match
Foo\B	the foo	No match
	the afoo test	No match
	the afootest	match

* \S - This character matches white spaces present in the string.
→ It is also equivalent to [\t, \n, \r, \f, \v]

Expression	String	Matched?
\S	Python	1 match
	RegEx	1 match
	PythonRegEx	No match

* '\d' - It matches the decimal numbers in the given string.

Expression	String	Matched?
\d	123de4	4 matches
	@5abc6	2 matches
	Python	No match

* '\D' - It matches all the non-decimal digits

Expression	String	Matched?
\D	123de4	2 matches
	@5abc6	4 matches
	Python	6 matches

* '\w' - Matches any alphanumeric character that includes alphabets, numbers & '_')

Eg

Expression	String	Matched?
	12&:;C	3 matches (at 12&:;C)
\W	%">!	No match

* '\W' - Matches any non-alphanumeric character

Expression	String	Matched?
	12&%C	1 Match (at 12&%C)
\W	Python	No match

Dt-14-03-2023

RegEx functions:-

→ The regular expression also provide functions that can be used for searching a string.

→ The different functions provided are
i) findall ii) search iii) split iv) sub

1) re.findall():-

→ This method returns a list of string containing all the matches for the search pattern.

```
# Program to extract numbers from a string.
import re
string = 'hello 12 hi 89, Howdy 34'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
#output : ['12', '89', '34']
```

#Program to extract specific word from the string

```
import re
txt = "The rain in Spain"
x = re.findall("ai", txt)
Print(x):
O/P = ['ai', 'ai']
```

iii) re.split() :-
→This method will split the string when the given patter matches & returns a list of string containing the split
#

```
import re
String = "Twelve : 12 Eighty nine : 89."
Patter = '\d+'
result = re.split(pattern, string)
Print(result) # O/P = "Twelve", "Eightynine"]
```

iii) re.sub() :-
→This method will create substitute for a pattern when it matches with the string & substitutes with the new pattern.
→This method will take 3 arguments i) Pattern
ii) Replace pattern
iii) String.
→We can use this method in the following way
#Program to remove all white-spaces

```
import re
#multiline string
string = 'abc 12\
de23\nf456'
string : 'abc 12 \
de 23 \n f45 6'
```

#matches all whitespace characters
Pattern = '\s+'
#empty string
replace = '~'
new_string = re.sub(Pattern, replace, string)
Print(new_string)
O/P
abc~12de~23~f45~6
iv) re.search() :-
→ This method takes two arguments i) Pattern, ii) string
→ This method finds the first position of the pattern where it matches with the string.
→ If pattern is found then it returns the match object otherwise it returns None.
→ syntax: match = re.search(Pattern, str)
Ex
import re
string = "Python is fun"
#check if 'Python' is at the beginning
match = re.search('\APython', string)
if match:
  Print ("Pattern found inside the string")
else:
  Print("Pattern not found")
#output: Pattern found inside the string.
Note:-
We can modify the search patterns in this method by using meta characters in the pattern provided if required.
Ex2 import re
text = "The rain in Spain"
x = re.search("Portugal", text)
print(x) O/P None

Exception Handling :-

Python exceptions :-

→ Exceptions can be categorized into the following types & they are

1) Zero Division Exception
2) Name error
3) Indentation Error
4) Type Error
5) IO Error.

→ The exceptions can be defined as abnormal condition or statement due which the program results in discreption of flow of execution.

→ In order to maintain the execution flow we should handel the exceptions that may generate in a program.

→ Exceptions are generated during the execution or detected at runtime. So inorder to handel them we must include exception handling blocks in our program which will execute if exception occurs.

→ We can hadel the exceptions in python by using the 3 different blocks 1) Try block
2) except block
3) finally block

Exception handling by using try & except block :-

→ We can use try & except blocks to handel the exception generated.

→ The try block contains all the code that may generate exception.

→ The except block contains the code that will execute if the exception provided in the try block is generated

→ In order to handel multiple exceptions we may write multiple except blocks.

Syntax:

```
try:
    Suspicious code
except exception1:
    block1 code
except exception2:
    block2 code
#WAP to handel
except exceptionn:
    blockn code
```

#WAP to handel to zero division error

```
try:
    x=int(input("Enter the first number"))
    y=int(input("Enter the second number:"))
    z=(x/y)
    Print("a/b = %d", %z)
except ZeroDivisionError:
    Print("Cannot divide by zero")
else:
    Print("Division successful")
Print("end of program")
```

O/P Enter the first number: 10
Enter the second number: 2
a/b=5
division successful
end of program

Enter the 1st number:10
Enter the 2nd number:0
Cannot divide by zero
end of Program

Finally block:-
→ Finally block will execute regardless of errors.
→ It is an optional block which executes after try or after except block.
→ It is generally used for representing external resources like file operations, database operations etc.
→ The finally block is always written after the except blocks.
ex
#Example of IOError.

```
try:
    Filepth = open("File1.txt", "r")
except IOError:
    Print("The file is not found")
else:
    Print("The file opened successfully")
    filepth.close()
finally:
    Print("I am finally")
Print("end of program")
```

~~Userdefine~~ Python Userdefined Exceptions:-

→ The exceptions that are defined as raised by the user are known as Userdefined exceptions.
→ We can define this exceptions by creating a new class inheriting from exception class.
→ The User defined exceeception are going to the raised by the keyword raised

Re Syntax:
raise exception_class(value)

→ ~~Raise~~ we can defined the exceptions according to the user requirement in the following way,

Eg

```
Class error(Exception):
  Pass
Class Derived_Smallvalue(error):
  Pass
Class Derived_largevalue(error):
  Pass
n=10
While True:
  try:
    i=int(input("Enter no "))
    if(i<n):
      raise Derived_smallvalue
    elif(i>n):
      raise Derived_largevalue
    break
  except Derived_smallvalue:
    Print("It is less than 10")
  except Derived_largevalue:
    Print("It is greater than 10")
Print("Congrats, it is ",10)
```

O/P

```
enter no: 11
It is greater than 10
```

Note The execution of user defined exceptions starts from the raised block then except block is executed which calls the user define exception class.

Date 21/05/2023
Page

Multi Threading:-

→ Multithreading represents execution of multiple threads simultaneously.
→ One thread represents one process that means multi-threading will hold more than one process at a time.
→ In systems multithreading is also referred as multi-tasking.

Multitasking:-

→ Performing more than one task simultaneously in one system is known as multitasking.
→ According to the type of task multitasking can be divided into two types i) Process based multitasking
ii) Thread based multitasking

i) Process based multitasking:-

→ Executing multiple tasks simultaneously where each task is a separate independent process is called process based multitasking.
→ This type of multitasking is handeled at os level.

ii) Thread based multitasking:-

→ Executing several tasks simultaneously where each tasks represent an independent part of the same program is known as thread based multitasking.
→ In this type of multitasking we will use a single program to control all the threads sequentially.
→ The program will be divided into multiple sub-programs or known as threads.
→ Thread based ~~multithreads~~ multitasking will be used to improve the performance of the system by reduce the response time.
⇒

Application areas of Multithreading →

→ We can apply the concept of multithreading in the following areas:

1) To implement Multimedia graphics.
2) To develop animations.
3) To develop video games.
4) To develop web and application servers, etc.

How to

Creating Threads: -

→ In order to create threads python provides one inbuild module called "threading"

Ex import threading.

→ After importing the module we can give the default thread that will hold every executing python program known as main thread to execute the current program.

Ex:

```
import threading
Print("Current Executing Thread:", threading.
                    current_thread().getname())
```

→ we can create user defined threads by using three different ways.

1) Creating a Thread using any class.
2) Creating a thread by extending Thread class
3) Creating a thread without extending Thread class

Creating a Thread:

1) Using any class:

→ The 1st way we are going to create the thread by directly importing the contents of threading module into program.

→ For the program no classes will be defined and the contents of threading will be directly accessed by the thread object anywhere in the Program.

ex:

```
From threading import *
def display():
    for i in range(1,11):
        Print("Child Thread")
t= Thread(target=display)
t.start()
For i in range(1,11):
    Print("Main Thread")
```

Note:

1. The above program will print the main thread & child thread message 10 times each.
2. The sequence of execution completely depends on the System, so we cannot guarrenty the exact output.

2) Extending Thread class:-

→ We can also create threads by extending the predefined thread class.

→ The thread class is present inside the threading module.

* → After extending the thread class we will use the object referenced of our class as thread object

Ex:

```
From threading import *
class MyThread(Thread):
    def run(self):
        for i in range(10):
            Print("Child Thread-1")
t=MyThread()
t.start()
for i in range(10):
    Print("Main Thread-1")
```

3) Without extending thread class:-

→ We can also create threads by using thread class object that will be created using the user defined class object without extending the thread class.

→ In this way we can make some parts of the program as threaded & some parts as non-threaded program.

*→ In order to do this while creating the thread object we can pass the object of the user defined class.

Ex:

```
from threading import *
class Test:
    def display(self):
        for i in range(10):
            Print("Child Thread-2")
obj=Test()
t=Thread(target=obj.display)
t.start()
for i in range(10):
    Print("Main Thread-2")
```

Without multithreading! –

→ When a program is not using multithreading then it will work according to the functions & the execution of the program depends on the function call.

→ According to the function call & the contents of the function, the program will execute & generate results.

Eg:

```
From threading import *
import time
def doubles(numbers):
    for n in numbers:
        time.sleep(1)
        Print("Double:", 2*n)
def Squares(numbers):
    For n in numbers:
        time.sleep(1)
        Print("Square:", n*n)
numbers = [1,2,3,4,5,6]
begintime = time.time()
doubles(numbers)
Squares(numbers)
Print("The total time taken:", time.time() - begintime)
```

With multithreading:–

→ One normal program can be converted into a multithreading program by any of the three ways.

→ When the program is converted into multithreading program then the response time of the program will be decreased by increasing the performance of the system.

→ Total time taken for the execution of the program will be reduced when we used multithreading.

eg:

```
From threading import *
import time
def doubles(numbers):
    for n in numbers:
        time.sleep(1)
        Print("Double", 2*n)
def Squares(numbers):
    for n in number
        time.sleep(1)
        Print("Square", n*n)
numbers = [1,2,3,4,5,6]
begintime = time.time()
t1 = Thread(target = doubles, args = (numbers,))
t2 = Thread(target = Squares, args = (numbers,))
t1.start()
t2.start()
t1.join()
t2.join()
Print("The total time taken:", time.time() - begintime)
```

Setting & Getting the name of a thread:-

→ Every thread in python has a name by default it is given by thread scheduler.

→ We can set the name of a thread according to our requirement.

eg:

```
t.getName = new name
```

→ We can also get the name of currently executing thread by using the method

t.getname() or t.getname

→ from threading import *

Setting & getting the name of a thread:-

→ Every thread in Python has a name default by given by thread scheduler.

→ We can set the name of a thread according to our requirement by using the method:
t.setName(newName)

→ We can also get the name of currently executing thread by using the method
t.getName()

eg
From threading import *
Print(current_thread().getName())
Current_thread().setName("Pawan Kalyan")
Print(current_thread().getName())
Print(current_thread().name)

O/P Main Thread
Pawan Kalyan
Pawan Kalyan

Note:-
Every Python thread has a name variable that also represents the name of the thread.

Thread numbering or Thread identification:-

Thread identification is done by identification numbers provided by thread scheduler

→ The numbers are identified by a variable called "ident".

→ Every thread will get a unique number and which will be in use until the total program terminates.

eg

```
from threading import *
def test():
    Print("Child Thread")
t = Thread(target = test)
t.start
Print("main Thread identification Number:", Current-thread().ident)
Print("child Thread identification Number:", t.ident)
```

O/P: child Thread
main Thread identification Number: 2492
Child Thread identification Number: 2768

active_count():- Dt: 23-03-2023

→ This method is used to count and return the no. of currently executing threads.

→ The method can be used in the following way

eg:

```
From threading import *
import time
def display():
    Print(Current_thread().getName(), "...started")
    time.sleep(3)
    Print(Current_thread().getName(), "... ended")
Print("The Number of active Threads:", active_count())
```

t1=Thread(target=display, name="childThread1")
t2=Thread(target=display, name="childThread2")
t3=Thread(target=display, name="childThread3")
t1.start()
t2.start()
t3.start()
Print("The Number of active Threads:", active_count())
time.sleep(5)
Print("The Number of active Threads:", active_count())

output

The Number of active Threads: 1
ChildThread1 ---. started
childThread2 --- started
childThread3 --- started
The Number of active Threads: 3
childThread1 --- ended
childThread2 --- ended
childThread3 --- ended
The Number of active Thread: 1

enumerate() function:-

- This function will return a list of all the currently executing threads
- This function will return the names of the threads in form of a list
→ we can use this function as

eg

```
from threading import*
import time
def display():
    Print(current_thread().getName(), "...started")
    time.sleep(3)
    Print(current_thread().getName(), "...ended")
```

t1=Thread(target=display, name="ChildThread1")
t2=Thread(target=display, name="ChildThread2")
t3=Thread(target=display, name="ChildThread3")
t1.start()
t2.start()
t3.start()
l=enumerate()
for t in l:
    Print("Thread Name:", t.name)
time.sleep(5)
l=enumerate()
for t in l:
    Print("Thread Name", t.name)

Output

ChildThread1 ---Started
ChildThread2 ---Started
ChildThread3 ---Started
Thread Name: MainThread
Thread Name: ChildThread1
Thread Name: ChildThread2
Thread Name: ChildThread2
ChildThread1: ---ended
ChildThread2: ---ended
ChildThread3: ---ended
Thread Name: MainThread

Note

When the thread execution is over after the ended of display() method all the threads will be taken out of currently active thread pool. ~~only the main thread~~
Only the main thread remains in execution from the starting of the ~~thread~~ program until the Program terminates.

isAlive():
→ This method check & returns the boolean value if the thread is currently active or not.
→ We can use this method in the following way
eg
From line 1 to 6 take from previous program
t1: Thread(target = display, name = "ChildThread1")
t2: Thread(target = display, name = "ChildThread2")
# t3: Thread(target = display, name = "ChildThread3")
t1.start()
t2.start()
Print(t1.name, "is Alive:", t1.isAlive())
Print(t2.name, "is Alive:", t2.isAlive())
time.sleep(5)
Print(t1.name, "is Alive:", t1.isAlive())
Print(t2.name, "is Alive:", t2.isAlive())
output
ChildThread 1 -- started
Child Thread 2 -- started
ChildThread1 is Alive: True
ChildThread2 is Alive: True
ChildThread 1 -- ended
ChildThread 2 -- ended
ChildThread1 is Alive: False
ChildThread is Alive: False

Join() method:-
→ This method will put a thread into temporary waiting states unless all other threads execute this thread will not be allowed to execute.

→ The join method will execute in the following way

eg

```
from threading import *
import time
def display():
    for i in range(10):
        Print("Seetha Thread")
        time.sleep(2)
t=Thread(target=display)
t.start()
t.join() # This line executed by main Thread
for i in range(10):
    Print("Rama Thread")
```

Output

Seetha Thread (10 times)
Rama Thread (10 times)

Note:

So the above Program the child thread executes first completely then allowing the main thread to execute.

join(seconds)

→ This method will allow the ~~method~~ thread to join the execution part after the specified time.

eg

```
t.join(5)
```

~~Daemon Threads~~ -

Daemon Threads:-

→ Daemon Thread is a thread which runs in the background.

→ The purpose of Daemon thread is to provide support to non-Daemon threads like main

thread & other child Threads
→ Example of the Daemon thread is Garbage Collector.
→ When the main thread runs will low memory then PVM calls this daemon thread to free up memory spaces.
→ In order to check a thread is Daemon thread or not we can use the method as t.isDaemon() which will return true or false.
→ we can also make a non Daemon thread to work as a Daemon thread by using the following method!

Eg: t.setDaemon(True)

→ The Daemon method can be created before starting a thread otherwise it will return error i.e RuntimeException: cannot set daemon status of active thread.

Eg:
```
From threading import *
Print(current_thread().isDaemon()) # false
Current_thread().setDaemon(True)
```
Output: RuntimeError: Cannot set daemon status of active thread

Note:-
We cannot set the current thread as daemon in the above program because the current thread is the main thread which cannot be made daemon.

Default Nature of Main Thread:-

→ By default the main thread is always non-Daemon but for the remaining threads defining by the user we can apply the daemon nature.

→ The reason for not allowing main thread to be daemon, as the main thread works as the parent thread for all the user defined threads & if the main thread is daemon then all other threads will become automatically Daemon threads.

WAP to create a User Defined thread & make it Daemon thread:

```
from threading import *
def job():
    Print("Child Thread")
t=Thread(target=job)
Print(t.isDaemon()) #false
t.setDaemon(True)
Print(t.isDaemon()) #True
```

Synchronization:-

→ If multiple threads are executing simultaneously then there will be a problem of data inconsistency.

→ In order to solve the data inconsistency Problem we can use the concept of Synchronization.

→→ eg

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

```
def wish(name):
    for i in range(10):
        print("Good Evening:", end=" ")
        time.sleep(2)
        print(name)
t1 = Thread(target=wish, args=("Dhoni",))
t2 = Thread(target=wish, args=("yuvraj",))
t1.start()
t2.start()
```

Output

Good Evening: Good Evening: yuvraj
Dhoni
Good Evening: Good Evening: yuvraj
Dhoni

→ In the above example the output is not generated appropriately for the two threads due to the problem of data inconsistency.

→ To overcome this problem we can use the concept of synchronization which will allow only one thread to execute at a time by locking the resources & thread.

— we can perform synchronization by using three different ways.

1. LOCK
2. RLOCK
3. Semaphore

Synchronization using:-

1) using LOCK:- complementing

→ Lock is the simplest way of synchronization where the thread is given a lock object which will control the execution of threads.

→ In order to create the lock object, we will call the lock() method
→ l=Lock()
→ After creating the lock object each thread will be allowed to acquire the lock at a time by calling the acquire() method
eg: l.acquire()
→ Once the thread completes its execution it will release the lock by calling release method.
eg: l.release();
→ The acquire & release process will continue for each thread separately
eg:

```
from threading import *
import time
l=Lock()
def wish(name):
    l.acquire()
    for i in range(10):
        print("Good Evening", end=" ")
        time.sleep(2)
    print(name)
    l.release()
t1=Thread(target=wish, args=("Dhoni",))
t2=Thread(target=wish, args=("Yuvraj",))
t3=Thread(target=wish, args=("Kohli",))
t1.start()
t2.start()
t3.start()
```

Problem with simple lock:-
→ In the simple lock we cannot share the resources i.e. one thread is holding the lock & other thread requests for it then the new thread i.e. the new thread that is requesting will be locked if it is the same thread then it will also wait it will be blocked

Date
Page

1) RLOCK:- Problem with Simple lock:-
In Simple lock we cannot share the resource ie, one thread is holding the lock & another thread is requesting but it. Then the new thread i.e, the new thread that requested will be block it if it is the same thread then also it will be blocked.

2) RLOCK:-
→ In this we will use RLock(Reentrant lock) to solve the problems in simple lock.
→ In this kind of locking mechanism the thread that holds the lock can acquire it multiple times.
→ If the lock is aquired by other thread then the thread will enter to block it if it requires the lock.

Example:-
```
From threading import *
l=RLock()
print("Main Thread trying to acquire lock")
l.acquire()
print("Main Thread trying to acquire Lock Again")
l.acquire()
```

Note:
Here the main thread will not be block & it can acquire the lock multiple times.

3) Semaphore:- Dt-24/03/2023
- We can also perform Synchronization using Semaphores
→ The lock & Rlock provides synchronization using one thread at a time to execute.

→ But in semaphore we can increase or decrease the no. of threads by allowing them to execute with shared resources.

→ The limitation to the no. of threads depend on the capacity of shared resource.

→

Creating semaphore object & allowing multiple threads to execute:-

Step 1: Create the object of Semaphore
S = Semaphore(counter)
→ The counter specifies the maximum number of threads allowed to execute simultaneously: default value is 1.
Step 2: Pass the acquire() method to the objects of semaphore which will provide access to one thread. One acquire method will decrease the value of counter by 1
Step 3: After the resource is used call the release method which will increase the value of counter by 1
Step 4: The semaphore object can be created in two different ways.

Way-1) S = Semaphore()
2) S = Semaphore(3)

WAP to use multithreading using Semaphore.

```
from threading import *
import time
s = Semaphore(2)
def wish(name):
    s.acquire()
    for i in range(10):
```

```
        Print("Good evening:", end=" ")
        time.sleep(2)
        Print(name)
        s.release()
t1=Thread(target="wish", args=("Virat",))
t2=Thread(target=wish, args=("Dhoni",))
t3=Thread(target=wish, args=("Yuvraj",))
t4=Thread(target=wish, args=("Rohit",))
t5=Thread(target=wish, args=("Surya",))
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

Note

On the above program out of the five threads two threads will be allowed to execute simultaneously through semaphore.

LOCK	RLOCK	Semaphore
* i) At a time only one thread can execute	* i) At a time only one thread can execute	* i) At a time multiple threads can execute
* ii) Same thread can't acquire the lock multiple times	* ii) Same thread can acquire the lock multiple times	* ii) multiple threads can acquire the lock multiple times
* iii) Lock concept is suitable for single or straight forward execution	* iii) This is suitable for nested or recursive execution	* iii) It can be used for straight & recursive executions
* iv) Sharing of resources are not allowed	* iv) Sharing of resources are not allowed	* iv) Sharing of resources are allowed

Date
Page

Inter Thread Communication :- (ITC)

→ When information sharing is allowed in multithreaded programs then we will see to ITC.

→ The ITC can be implemented by the use of join() & sleep() method.

→ The join() method will allowed calls one thread to execute. when other threads finish their execution

→ The sleep(). method puts the current thread into inactive state & calls the other thread to execute. This represent the ITC where one thread is responsible for the execution of other thread.

Graphics Programming Using Python:-

→ We can develop GUI Programs in python by using 3 different types of UIs & they are.

- Tkinter -
- wxPython
- JPython.

→ The three different UIs are used for developing GUI programs using the Tk toolkit provided for Tkinter UIs, wx windows for developing wxPython UIs & another toolkit for combining java with python to develope JPython UIs

Developing UI Using Tkinter module:-

→ In order to develope user interfaces we can use the Tkinter GUI toolkit by writing

i) import Tkinter.

→ After importing the module we can create a blank interface or frame by writing

ii) top = tkinter.Tk()
top.mainloop()

Output

Tkinter widgets:-

→ These are the components that are used to represent various operation done inside the frame or window.

— The different widgets provided by tkinter are
1) Button :-
→ This widgets is used to display the button & to operate the click operation
2) Canvas :-
— This widgets is used to get a white screen where we can draw different shapes.
3) Checkbutton :-
— This widget will create check boxes where we can select one or multiple ~~options~~ boxes.
4) Entry :-
— This widget will be used to enter a single line text
5) Frame :-
— This widget is used to create a container which will contain other widgets & we can organised the widgets according to the frame
6) Label :-
— This widget is used to display some text to the user. We can also use images on it
7) Listbox :-
— This widget is used to provide a number of list options to the user from which the user can select one option.
8) Menbutton :-
— This widget is used to display various menus to the user
9) Menu :-
— This widget provides various commands to the user which are used inside menubutton

9. Message! -
- This widget is used to create a multiline text field where user can insert multiple lines of text.

10. Radiobutton! -
- This widget is used to create radiobuttons out of which only one option can be selected at a time.

11. Scale! -
- The Scale widget is used to provide a slider

13. Scrollbar! -
- This widgets is used to add scrolling option to other widgets.

14. Text! -
- This widget is used to display text in multiple lines.

15. TOplevel! -
- This widget will provide a seperate window container.

16. Spinbox! -
- This widget is modified by variant of entry ~~widget~~ widget which can be used to enter single line text.

17. Panedwidow! -
- This widow container can contain different pane arranged horizontally or vertically.

18. Label frame! -
- This widget is a container which acts as a spacer both complex window layouts.

19. tkmessage box! -
This widgets is used to messages boxes in your application.

Creating Buttons:-

1) To add a button to the interface we can write the following syntax -

W: Button(master, option: values)

- The master attribute will add the button into the parent widow
- The Options represents the various attributes of a button
- We can use multiple options separated by commas,
- Some of the attributes are

1) activebackground: to set the background colour when button is under the cursor.
2) activeforeground: to set the foreground colour when button is under the cursor.
3) bg: to set the normal background color.
4) Command: to call a function.
5) font: to set the font on the button label.
6) image: to set the image on the button.
7) width: to set the width of the button.
8) height: to set the height of the button.

```
import tkinter as tk.
r = tk.Tk()
r.title("Counting Seconds")
button = tk.Button(r, text='Stop', width=25, command
button.pack()                                =r.destroy)
r.mainloop()
```

2. Canvas:-
→ Syntax:- W = Canvas(master, option = value)

- The canvas contains the following options that can be used for changing the format of it.

bd: to set the border width in pixels.
bg: to set the normal background colour.
cursor: to set the cursor used in the canvas.
highlightcolour: to set the colour shown in the focus highlight.
width: to set the width of the widget.
height: to set the height of the widget.

```
from tkinter import *
master = Tk()
w = canvas(master, width=40, height=60)
w.pack()
canvas_height = 20
canvas_width = 200
y = int(canvas_height/2)
w.create_line(0, y, canvas_width, y)
mainloop()
```

3. CheckButton:-
→ Syntax:- CheckButton(master, option = value)
@
→ • The checkbutton contains the following options that can be used for changing the format of it.
• Title:- To set the title of the width.
• activebackground:- to set the background color when widget is under the cursor.

• activeforeground: to set the foreground color when widget is under the cursor.
• bg: to set the normal background.
• command: to call a function.
• font: to set the font on the button label.
• image: to set the image on the widget.

```
From tkinter import *
master = Tk()
var1 = IntVar()
Checkbutton(master, text='male', variable=var1).
.grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text='female', variable=var2)
.grid(row=1, sticky=W)
mainloop()
```

4. Entry' -

→ Syntax: Entry(master, option=value)

• bd:

— Entry contains the following options that can be used for changing the format of it

1) bd
2) bg
3) cursor
4) command
5) highlightcolor:
6) width
7) height

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

5. Frame :-

→ Syntax :- Frame(master, option = value)

→ There are number of options present in the frame that are used for changing the format of it

- highlightcolor, bd, bg, cursor, width, height

```
from tkinter import *
root = Tk()
frame.pack()
bottomframe = Frame(root)
bottomframe.pack(side=BOTTOM)
redbutton = Button(frame, text='Red', fg='red')
redbutton.pack(side=LEFT)
greenbutton = Button(frame, text='Green', fg='green')
greenbutton.pack(side=LEFT)
bluebutton = Button(frame, text='Blue', fg='blue')
bluebutton.pack(side=LEFT)
blackbutton = Button(frame, text='Black', fg='black')
blackbutton.pack(side=BOTTOM)
root.mainloop()
```

6. Label :-
→ Syntax :- W = Label (master, option = value)

• bg, fg, command, font, image, width & height

```
from tkinter import *
root = Tk()
w = Label(root, text="GeeksforGeeks.org!")
w.pack()
root.mainloop().
```

7. Listbox :-
→ Syntax :- W = Listbox (master, option = value)

• highlightcolor, bg, bd, font, image, width, height

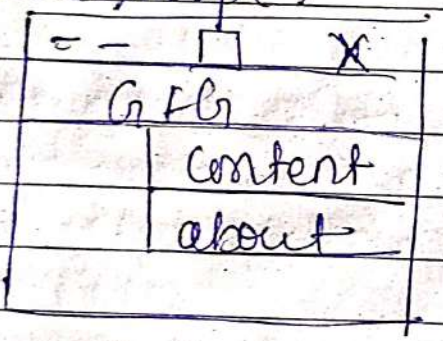
```
from tkinter import *
top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```

8. MenuButton :-
→ Syntax :- W = MenuButton (master, option = value)

• activebackground, activeforeground, bg, bd, cursor, image, width, height, highlightcolor.

```
from tkinter import *
top=Tk()
mb=Menubutton
mb=Menubutton(top, text="GfG")
mb.grid()
mb.menu=Menu(mb, tearoff=0)
mb["menu"]=mb.menu
cVar=IntVar()
aVar=IntVar()
mb.menu.add_checkbutton(label='Content',
                        variable=cVar)
mb.menu.add_checkbutton(label='About',
                        variable=aVar)
mb.pack()
top.mainloop()
```

O/P



10. Message :-

→ Syntax : w=Message(master, option=value)

bd, bg, font, image, width, height

```
from tkinter import *
main=Tk()
ourMessage='This is our message'
messageVar=Message(main, text=ourMessage)
messageVar.config(bg='lightgreen')
messageVar.pack()
main.mainloop()
```

11. Radio Button! –

→ Syntax: RadioButton(master, option=value)

Dialogue & Message Boxes! –

→ Python provides different types of message boxes according to their requirement & use.

→ we have the following kinds of message boxes to display i) simple message, 2) showing, 3) opening other windows for selecting different options.

~~Dialog box!~~ –

~~Dialogue box!~~ – Message box! –

→ The dialog box is created to show error message or simple display messages which can be created by using the following methods.

i) showerror(...) – This will show an error dialogue box.

ii) showwarning(...) – This would display a warning dialogue box.

iii) showinfo(...) – It will display a normal message box.

Dt-31-03-2022

Method for Message box! –

→ We can have the different ~~kinds of~~ types of methods to control the execution of the message boxes & they are

- askokcancel(title=None, message=None, **options)
Ask if operation should proceed; return true if the answer is ok.
- askquestion(title=None, message=None, **options)
Ask a question.

• askretrycancel(title=None, message=None, **options)
Ask if operation should be retried; return true if the answer is true.
• askyesno(title=None, Message=None, **options)
Ask a question; return true if the answer is yes.
• askyesnocancel(title=None, message=None, **options)
Ask a question; return true if the answer is yes, None if cancelled
• showerror(title=None, message=None, **options)
Show an error message.
• showinfo(title=None, message=None, **options)
Show an info message.
• showwarning(title=None, message=None, **options)
Show a warning message.

Dialogue box :-

→ It will open a separate window for different other options like opening a folder, changing colours etc.

→ In order to open a file dialogue box we can use the following code.

```
from tkinter import *

def callback():
    name = askopenfilename()
    print(name)
errmsg = 'Error!'
Button(text='File Open', command=callback).pack(fill=X)
mainloop()
```

Note

The look & title of dialogue boxes depends on the operating system of the users according to that the dialogue box will look different.

Opening Color choosen dialogue box:-

→ Using the method for color choosen

result = tkColorChooser.askColor(color, option = value, ...)

we can open the color choosen dialogue box,

→ This will provide options to choose color from different shades of ~~over~~ RGB

→ The option provides different values like Title & parent, To provide the title & parent window in which color chooser will be attached.

→ we can create a color chooser window by writing

```
from tkinter import *

def callback():
    result = askColor(color="#6A9662", title="Bernd's
                      Color Chooser")
    Print(result)
root = Tk()
Button(root, text='Choose color', fg='darkgreen',
       command=callback).pack(side=LEFT, padx=10)
Button(text='Quit', command=root.quit, fg="red").
                    pack(side=LEFT, padx=10)
mainloop()
```

Note

every color chooser widow will have a parent component attached to it. In the previous example button1 acts as the parent for color chooser window.

Layout Managers:-

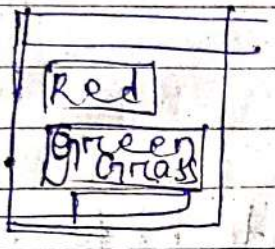
→ The layout manager is used to arrange the components according to the window or frame.
→ Python provides 3 kinds of layout managers & they are
1) Pack
2) grid
3) Place
→ The layout managers are basicially used for the following purposes
i) arrange widgets on the screen
ii) registering those widgets with the window system.
iii) managing the widgets according to the screen size.

i) Pack :-

→ This layout manager is used to restruct the size of the window according to the contents of the widgets.
→ According to the type & no. of widget the window will be automatically arranged.
→ The pack method should be used for each individual widget

```
from tkinter import *
root = Tk()
Label(root, text = "Red Sun", bg = "red", fg = "white").Pack()
Label( root, text = "Green Grass", bg = "green", fg = "black")
                                                    .Pack()
Label( root, text = "Blue sky", bg = "blue", fg = "white")
mainloop().                                         .Pack()
```

O/P



Options for Pack method:-

Fill Option:-

→ This attribute represents how to fill the components or widgets in the window.

→ According to the value provided for fill we can arrange the components in the window.

Eg:
```
from tkinter import *
root = Tk()
w = Label( root, text = 'Red sun', bg = 'red', fg = 'white')
w.Pack(Fill = X)
w = Label( root, text = 'Green Grass', bg = 'green', fg = 'black')
w.pack(Fill = X)
mainloop()
```

→ we can provide three options in fill ~~X, Y, BOTH~~

X, Y, BOTH

X → HORIZONTAL

Y → VERTICAL

BOTH → HORIZONAL & VERTICAL BOTH

Padding:-
→ This attribute is used to fill the components with padding in both the directions
→ The values that can be given for padding are
Padx
Pady
ex.
i) W.Pack(Fill=X, Padx=10)
ii) W.Pack(Fill=X, Pady=10)

Internal padding:-
→ We can change the arrangement between the component boundary & its content by using internal padding.
→ The internal paddings are represented by ipadx & ipady
→ ipadx will change the space in horizontal component boundary
→ ipady will change the space between vertical component boundary.

Placing widgets side by side:-
- We can place the widget side by side by using side attribute
- The value for side attribute can be LEFT or RIGHT

Place Geometry Manager
- Geometry manager allows to explicitly place the widget at different positions.
- The positions can be given by using a method "geometry()" that will change the size & position

Ob wrdown.
Syntax:- geometry(width x height + x_offset + y_offset)

Note:-
we need place method to place each widget inside the root window & the syntax for place:-
place(x_offset, y_offset, width, height)

Grid Manager:-
- The grid manager is used to arrange the components in Grid i.e. in Tabular format.
- we can manually set the Grid ie the no. of rows & columns using the attributes rows & columns in grid().
- According to the provided value a two dimensional table will be created & the widget will be placed according to their row & column values.
- If two widgets have same row value then they will be placed in a single row & if two widgets have same column value then the widget will be place in same column.

Turtle Programming using turtle Module.

→ This module is used to draw different shapes in python

→ The turtle module provides different methods to control the movement of the turtle on the drawing board.

→ The different methods used for turtle movement are

METHOD	PARAMETER	DESCRIPTION
Turtle()	None	Creates & returns a new turtle object
forward()	Amount	Moves the turtle forward by the specified amount
backward()	Amount	Moves the turtle backward by the specified amount
right()	Angle	Turns the turtle clockwise
left()	Angle	Turns the turtle counter clockwise
penup() or up()	None	Picks up the turtle's pen
pendown() or down()	None	Puts down the turtle's pen
color()	Color name	changes the color of the turtle's pen
fillcolor()	Color name	changes the color of the turtle will use to fill a polygon
heading()	None	Returns the current heading
position()	None	Returns the current position
goto()	any	Move the turtle to position arg
begin_fill()	None	Remember the starting point for a filled polygon
end_fill()	None	close the polygon and fill with the current fill color

dot()	None	Leave the dot at current position
stamp()	None	Leaves an impression of a turtle shape at the current location
shape()	shapename	should be 'arrow', 'classic', 'turtle' or 'circle'

Steps for executing turtle program:-

→ There are four basic steps that can be used to execute turtle program and they are.

1. Import the turtle module,
2. Create the turtle object by using Turtle() method.
3. Call the turtle movement methods to draw the shapes.
4. Run turtle.done() method to complete the drawing process.

Draw a star shapes using turtle.