Core Python

Introduction to Programming

Q). What is a program?

Sol).

A program is a set of instructions given to a computer to perform a specific operation.

- While executing the program, raw data is processed into a desired output format. These computer programs are written in a programming language which are high level languages.
- High level languages are nearly human languages which are more complex than the computer understandable language which are called machine language, or low-level language.
- The computer only understands binary language (the language of 0's and 1's) also called machine-understandable language or low-level language but the programs we are going to write are in a high-level language which is almost similar to human language.

Q). What is Programming Language?

Sol).

- Programming Language is a tool/medium that allows us to interact with the computers.
- In other words, programming language helps us to communicate with computers.
- As we know that we understand simple languages (high level languages) but machine understands only machine languages (in the form of 0 and 1).
- Machine Language, Assembly Language

- c, c++, java, python, cobol, Fortran, c#, php, go, etc.

Q). types of languages?

Sol).

- **Low Level Languages-:** Languages which are dependent on machine are called low level languages. They directly interact with the hardware. they don't require any compiler or interpreter. (machine language and assembly language)
- Utility program (**Assembler**) is used to convert assembly code into executable machine code.
- **High Level Languages-:** Languages Which are independent on machine are called High level languages. compiler or interpreter is present here.
- High Level Programming Language are portable but require Interpretation or compiling to convert it into a machine language which is computer understood.
- The computer only understands binary language (the language of 0's and 1's) also called machine-understandable language or low-level language but the programs we are going to write are in a high-level language which is almost similar to human language.

Q). write down the Characteristics of a programming Languages?

Sol).

- A programming language must be simple, easy to learn and use, have good readability and human recognizable.
- Abstraction is a must-have Characteristics for a programming language in which ability to define the complex structure and then its degree of usability comes.
- A portable programming language is always preferred.
- Programming language's efficiency must be high so that it can be easily converted into a machine code and executed consumes little space in memory.
- A programming language should be well structured and documented so that it is suitable for application development.

- Necessary tools for development, debugging, testing, maintenance of a program must be provided by a programming language.
- A programming language should provide single environment known as Integrated Development Environment (IDE).
- A programming language must be consistent in terms of syntax and semantics.

Q). type of language, their developers and the year in which these languages have been developed is written below-:

Programming Languages	Developers	year
1. Plankalkul	Konrad Zuse	1943-1945
2. FORTRAN	John Backus	1956
3. LISP	John McCarthy	1958
4. COBOL	Grace Hopper	1959
5. BASIC	John Kemeny and Others	1964
6. SIMULA	Ole-john-Dahl and Kristen Nygaard	1965
7. Pascal	Nikolas Wirth	1971
8. c	Dennis Ritchie	1972
9. C++	Bjarne Stroustrap	1979
10. Python	Guido van Rossum	1991
11. R	Robert Gentalman	1993
11. java	James Gosling	1995

Important Questions-:

- First high-level language-: Plankalkul
- First procedure-oriented programming language-: FORTRAN
- FIRST OOPS PROGRAMMING LANGUAGE-: SIMULA

Introduction to python

Q). What is python?

Sol).

Python is an interpreted, object oriented, high level and general-purpose programming language. It is also a scripting language.

It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.

There are two major Python versions:

Python 2x and Python 3x.

Both are different.

But I suggest you to use above 3.5 because all industries are using above 3.5.

Q). write down the Features of Python? (important)

Sol).

- Very simple and straight forward syntax.
- It is Dynamically typed language (Data Type decide at run time).
- Indentation is used in place of curly braces.
- Use variable without declaration.
- Emphases on code readability.
- Automatic memory management & garbage collector.
- A broad standard Library.
- Interpreted language.
- Support of Multithreading.
- Multi paradigm programming language i.e., object-oriented programming, procedural programming.
- Platform independent.
- Interactive model (interactive testing and debugging of snippets of code)

- Portable (can run on variety of h/w platforms)
- Databases

Q). Why Python is so popular? (important)

Sol).

- First and popular answer is because of its rich library.
- As it uses simple English Words which make it beginners' language.
- Line of code is very less so it consumes the time of programmer.
- It is Very useful for machine learning and Deep learning. That is making it more popular.
- Maximum industries are moving towards python, which is making maximum programmers to learn the python.
- Last but not the least it is saving lot of time in writing codes. That is most precious thing.

Q). what are major applications that can be developed using python?

Sol)

- Web Applications
- Desktop GUI Applications
- Software Developments
- Scientific & Numeric Applications
- Business Applications
- Console Based Applications
- 3D CAD Applications
- Enterprise Applications

Q). Python has Library For?

Sol).

- Web Frameworks
- GUI (Graphical User Interface)
- Multimedia
- Database
- Networking
- Automation
- Image processing
- IOT (Internet of things)
- Scientific Computing
- Web Scraping
- System Administration

Q). How Python is interpreted?

Sol).

An **interpreter** is a kind of program that executes other programs. When you write **Python programs**, it converts source code written by the developer into **intermediate language** which is again translated into the native language / machine language that is executed.

The **python code** you write is compiled into python bytecode, which creates file with extension **.pyc** . The bytecode compilation happened internally, and almost completely hidden from developer. Compilation is simply a translation step, and byte code is a lower-level, and **platform-independent**, representation of your source code. Roughly, each of your source statements is translated into a group of byte code instructions. This byte code translation is performed to speed execution **byte code** can be run much quicker than the original source code statements.

The **.pyc file**, created in compilation step, is then executed by the Python virtual machines. The Virtual Machine just a big loop that iterates through your **byte code** instructions, one by one, to carry out their operations. The **Virtual**

Machine is the runtime engine of Python and it is always present as part of the Python system, and is the component that truly runs the **Python scripts**. Technically, it's just the last step of what is called the Python interpreter.

Q). What IDE to use for Python? (important)
Sol).

There is various IDE for Python development. Some popular IDEs are:

- PyCharm IDE
- Pydev IDE
- Wing IDE
- Eric Python IDE
- Vim IDE
- IPython Notebook

Q). What are all the operating system that Python can run on?
Sol).

Python is a platform independent language, it works for all Operating Systems like, Windows, Unix, Linus, MacOS etc.

Q). Is python a case sensitive language?

Sol). Yes, Python a case sensitive language

Case sensitive means that x is different from X. The variable of John is different from the variable of john.

Q). What are the drawbacks of Python? (important)
Sol).

Speed

Python is **slower** than C or C++. But of course, **Python** is a high-level language, unlike C or C++ it's not closer to hardware.

Keywords

Q) Define Keywords in python. Name all the Keywords in python.

Sol) keywords are the reserved words in Python.

33 Reserved Words

We cannot use a keyword as a variable name, function name or any other identifier.

They are used to define the syntax and structure of the Python language.

keywords are-:

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

i can also find out all keywords by using programs.

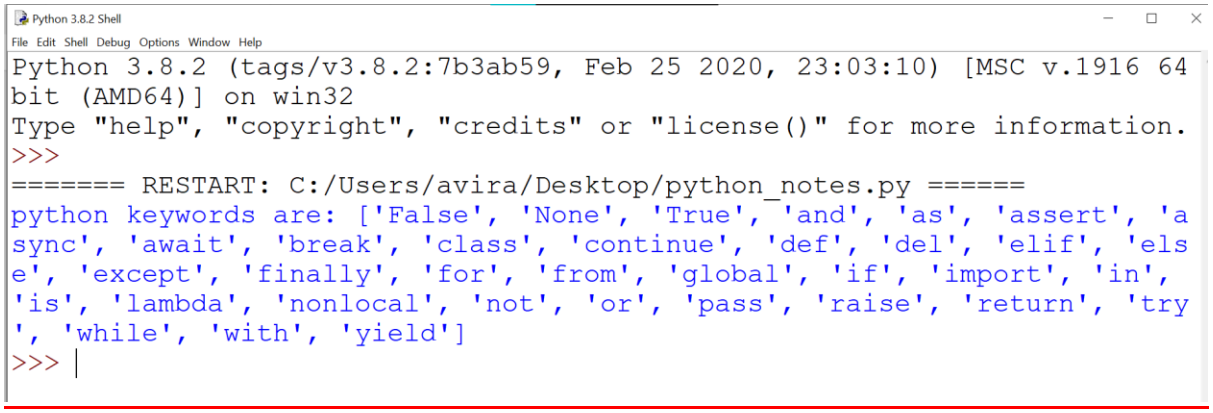
Q) Write a program that print all keywords in python? (important)

Sol).

File Edit Format Run Options Window Help

```
import keyword
print("python keywords are:", keyword.kwlist)
```

Output-:



Q). define variables? Show how we can declare variables?

Sol).

variables are a named location use to store data in the memory. variables are also called container that holds data which can be changed later throughout programming.

Various types for declaring a variable-:

a=10

a, b, c = 5,1.2,"hello"

a = b = c = "amazing python"

Data Types

Q) What is data type in python. Write the types of data type?

Sol).

Data is the type of data that you can store inside a variable.

As you know that every value is stored in some variable, so the type of values which you can store with a variable, is known as the data type of that variable.

Every value in Python has a datatype.

- There are 14 Data types in python.
- int (in python 2 long is also their but not in python 3)
- float (floating point values)
- Complex (for Scientific calculations or programs) (10+2j)
- bool
- str
- bytes

- bytearray
- range
- list
- tuple
- set
- frozenset
- dict
- None

- in python we need not to define name of data types that is why they are not keywords.
- in python everything is an object. so, data types are actually classes and variables are instance (object) of these classes.
- a=10
- id(a)----it will define the address of an object.

Q). Write 3 built-in data types in python and their functionality. (important)

Sol).

- **print()--:** it is use to print the value of objects.
- **type()**--: it is use to define the type of an object.
- **id()**--: it will return the address of an object.

File Edit Format Run Options Window Help

```
a = 10
print("value of a is:",a)
print("Data type of a is:",type)
print("memory location of a is",id(a))
```

Output

```
value of a is: 10
Data type of a is: <class 'type'>
memory location of a is 140706395641792
>>>
```

Q). describe the int data type in python?

Sol). int data type is used to store the data in the form of integer values.

we can specify it in various ways

1. Decimal form
2. Binary Form
3. Octal form
4. Hexadecimal form

Q). describe decimal, binary, octal and hexadecimal form in python.

Sol).

decimal-: (base-10) and numbers between (o to 9)

binary-: (base -2) and numbers between (0 and 1)

if any number starts with 0b or 0B, then this number is binary number (0b10110, 0B111101)

Ex).

```
a=0B1111
print(a)
```

octal-: (base-8) and numbers between (0-7)

if number starts with 0o or 0O then numbers will treat as octal number. (0o1111, 0O1111)

hexadecimal-: (base-16), numbers between (0-9) and symbols between (a-f, A-F).

numbers should be prefixed with 0x or 0X. (0XFace, 0xBee).

Q). list the method names that can convert the base of decimal, binary, octal and hexadecimal? (important)

Sol).

bin()-: to convert decimal/octal/hexadecimal values into binary.

oct()-: to convert decimal/binary/hexadecimal values into octal.

hex()-: to convert decimal/binary/octal values into hexadecimal.

```
File  Edit  Format  Run  Options  Window  Help
a = 20
print("a in binary form is:",bin(a))
print("a in octal form is:",oct(a))
print("a in hexadecimal form is:",hex(a))
```

Output-:

```
a in binary form is: 0b10100
a in octal form is: 0o24
a in hexadecimal form is: 0x14
>>>
```

python gives output only in decimal form.

max value or max size are not applicable for python.

Q). Explain **float data type** in python?

Sol).

if you want floating value, go for float data type. (no double type) these values can only be defined in decimal way.

```
File  Edit  Format  Run  Options  Window  Help
a = 20.675
print("value and type of a is:",a,type(a))
```

Output-:

```
value and type of a is: 20.675 <class 'float'>
>>>
```

Exponential form=

1.23e3 = 1.23*10^3

Q). Explain **complex data type** in python? (important)

Sol). complex data type is for storing the complex values in python.

format-: a+bj (where a is real part and b is imaginary part).in imaginary part it should only be j symbol.

j^2 = -1(where j is square root of -1)

Q). example of complex value and its type.

Sol).

```
File Edit Format Run Options Window Help
x= 10+20j
y=2.7+1.5j
print("value and type of x is:",x,type(x))
y = 10j
print("value and type of y is:",y,type(y))
```

Output-:

```
value and type of x is: (10+20j) <class 'complex'>
value and type of y is: 10j <class 'complex'>
>>>
```

Q). program to print real and imaginary form using methods.

Sol).

```
File Edit Format Run Options Window Help
x = 10+20j
y = 20+30j
print("real part of x is:",x.real)
print("imaginary part of x is:",x.imag)
print("real part of y is:",y.real)
print("imaginary part of y is:",y.imag)
```

Output-:

```
real part of x is: 10.0
imaginary part of x is: 20.0
real part of y is: 20.0
imaginary part of y is: 30.0
>>>
```

Q). Program to add, subtract, multiply and divide the two complex numbers? (important)

Sol).

File Edit Format Run Options Window Help

```
x = 10+5.6j
y = 5+7j
som = x+y
sub = x-y
mul = x*y
divi = x/y
print("Addition of x and y is:",som)
print("substraction of x and y is:",sub)
print("multiplication of x and y is:",mul)
print("Division of x and y is:",divi)
```

Output-:

```
Addition of x and y is: (15+12.6j)
substraction of x and y is: (5-1.4000000000000004j)
multiplication of x and y is: (10.800000000000004+98j)
Division of x and y is: (1.2054054054054055-0.5675675675675675j)
>>>
```

note-: real part can be of any integer type but imaginary part should be of decimal type.

you can also perform mathematical operations on complex numbers.

Q) describe the **bool data type** in python

Sol). bool data type stores the value in the form of True and False.

internally True is treated as 1 and False is treat as 0.

Q). program to compare two numbers with bool method?

Sol).

```
File  Edit  Format  Run  Options  Window  Help
x = 10
y = 15
res = bool(x==y)
print(res)
```

Output-:

```
False
>>>
```

Q). program that shows the operations in bool.

Sol).

```
x = 1
print(bool(x))      ## Output-: True
y = 0+1
print(bool(y))     # Output-: True
```

Q). Describe the **str data type** in python.

Sol).

any sequence of character enclosed with single or double quotes. but single quotes are recommended in python.

```
File  Edit  Format  Run  Options  Window  Help
x = "welcome to python programming!"
print(x)
y = '''hello how are you
        welcome to python programming!'''
print(y)
```

Output-:

```
welcome to python programming!
hello how are you
        welcome to python programming!
>>>
```

single quote or double quotes icon are used to define only single line string literals.

but if you want multiline string literal you can use triple single or double quotes.

x = '''This is

nexgen'''

You can print strings in this way is well-

"Hello everyone! Welcome to this 'Python' Series".

Q). show string concatenation with examples? (important)

Sol). when we use + operator between two or more strings then they will concatenate each other.

```
File  Edit  Format  Run  Options  Window  Help
print ("10"+"20"+"30")
print ("tarun"+"Chaudhary")
```

output

```
102030
tarunChaudhary
>>> 
```

Note-: you cannot concatenate a string with an integer.

Q). describe some important methods in str data type with example?

Sol).

Python has a set of built-in methods that you can use on strings.

1. input()-: input() is a string function that is used to take input from the user. As it is a function of string so it takes input in the form of string.

Ex-:

```
name = input("enter Your Name:")
print(name)
print("data type is:",type(name))
# but if you want to take input in any other form
# then you are required to typecast it.
num = int(input("Enter any  number:"))
print(num)
print("data type is:",type(num))
```

Output-:

```
enter Your Name:Avanish
Avanish
data type is: <class 'str'>
Enter any  number:1234
1234
data type is: <class 'int'>
>>>
```

---

2. len()-: it is a built function of str data type that is use to calculate the length of any string value

Ex-:

```
x = "Avanish Singh"
print("Length of x is:",len(x))
```

Output-:

```
Length of x is: 13
>>>
```

3. find( )-: it returns the lowest index in the string where the substring is found. and returns -1 on failure. (important)

Ex).

```
name = input("Enter a string: ")
sub_name = input("Enter the sub string: ")
position = name.find(sub_name)
print(sub_name, 'is found at', position)
```

Output-:

```
Enter a string: hello python hello py
Enter the sub string: ello
ello is found at 1
>>>
```

4. index( )-: similar to find but returns error if search not found. Whereas find return -1 if substring not found.

Ex).

```
name = input("Enter a string: ")
sub_name = input("Enter the sub string: ")
position = name.index(sub_name)
print(sub_name, 'is found at', position)
```

Output-:

```
Enter a string: python
Enter the sub string: a
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\python_notes.py", line 4, in <module>
    position = name.index(sub_name)
ValueError: substring not found
>>>
```

5. join( )-: it joins the elements of with every element to the end of the string.

Ex.)

```
data = 'python'
new_data = 'a'.join(data)
print(new_data)
```

Output-:

```
payatahaoan
>>>
```

6. split()-: this string method will divide the character from where you want to split.

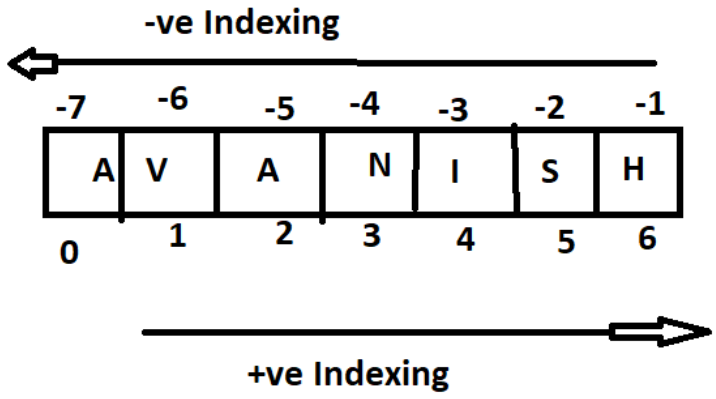
Ex-:

```
path = input("Enter any path: ")
res = path.split("/")[-1]
print("File Name is: ", res)
```

output

```
Enter any path: c:/python/first/demo.py
File Name is:  demo.py
>>>
```

Indexing in Python



Q). take a string and show some indexing examples?

Sol).

```
s = "Tarun Chaudhary"
print("elemnt on 0 index:",s[0])
print("elemnt on -1 index:",s[-1])
print("elemnt on -15 index:",s[-15])
print("elemnt on 3 index:",s[3])
print("elemnt on -3 index:",s[-3])
```

Output-:

```
elemnt on 0 index: T
elemnt on -1 index: y
elemnt on -15 index: T
elemnt on 3 index: u
elemnt on -3 index: a
>>>
```

Q). define **list data type** and its features?

Sol). list is a container that stores the heterogeneous elements.

- list is of a mutable type. because you can read as well as write the contents in list.
- it allows duplicate members.
- all fundamental datatypes are immutable.
- list is a data type where insertion order is preserved and duplicates are allowed.
- identification of list should be done by the index.

Ex-:

```
my_list = [10, "mca", 2+6j, 8299089703,3333.56, True]
print(my_list)
print(type(my_list))
```

Output-:

```
[10, 'mca', (2+6j), 8299089703, 3333.56, True]
<class 'list'>
>>>
```

Q). What is mutability concept in python? (important)

Sol). mutability/mutable =when your data container/data structure providing you the facility to make changes into your data then it is of mutable type. changes

into your container does not affect the memory location, it will remain same.

Q). What are the ways to create an empty list?

Sol). as you know that list is also a function. So, either you can use the function or you can use square brackets [] to create list.

li = list()

li = []

Q). program to extract the elements from the given list. (important)

Sol).

in python if you want to extract any element from the given list you have to extract them on the basis of index number.

Index number is always defined inside square brackets.

```
my_list = [10, "mca", 2+6j, 8299089703,3333.56, True]
print("2nd element of my list is:", my_list[3])
print("last element of my list is:",my_list[-1])
```

Output-:

```
2nd element of my list is: 8299089703
last element of my list is: True
>>>
```

Q). describe all the list methods with examples.

Sol).

1. append( )-: it will add a new element at the end of the list.

Ex).

```
# suppose i have an empty list
# and i want to insert some data into that
li = list()
li.append(1)
li.append("Avi")
li.append(17.89)
print(li)
```

outout-:

```
[1, 'Avi', 17.89]
>>>
```

2. extend( )-: it will append new list with current list.

Ex).

```
stu1_info = [10,'ravi', 25, 'MCA', 4534567877]
stu2_info = [11,'avi', 25, 'BCA', 5676767877]
stu1_info.extend(stu2_info)
print(stu1_info)
print(len(stu1_info))
```

Output-:

```
[10, 'ravi', 25, 'MCA', 4534567877, 11, 'avi', 25, 'BCA', 5676767877]
10
>>>
```

Ex) there are some other ways to merge two lists into one list.

```
stu1_info = [10,'ravi', 25, 'MCA', 4534567877]
stu2_info = [11,'avi', 25, 'BCA', 5676767877]
print("after concatnation:" , stu1_info + stu2_info)
stu1_info.append(stu2_info)
print(stu1_info)
```

Output-:

```
after concatnation: [10, 'ravi', 25, 'MCA', 4534567877, 11, 'avi', 25, '
BCA', 5676767877]
[10, 'ravi', 25, 'MCA', 4534567877, [11, 'avi', 25, 'BCA', 5676767877]]
>>>
```

3. index( )-: it will return index of the element if found.

Ex).

```
color = ['red', 'yellow', 'blue', 'black', 'white', 'brown']
color_name = input("Enter the color name you want to search: ")
res = color.index(color_name)
print(res)
```

Output-:

```
Enter the color name you want to search: white
4
>>>
```

count( )-: it will count existence of element in list. (important)

Ex).

```
color = ['red', 'yellow', 'blue', 'black', 'white', 'brown', 'black']
color_count = color.count('white')
print(color_count)
```

Ouput-:

```
1
>>>
```

4. reverse( )-: it will reverse a list.

Ex).

```
stu_info = [10,'abcd', 25, 'MCA', 4534567877, 'xyz']
stu_info.reverse()
print(stu_info)
```

Output-:

```
['xyz', 4534567877, 'MCA', 25, 'abcd', 10]
>>>
```

5. insert( )-: it will insert new element at specific position. where first parameter is index and second is element.

Ex).

first.py - C:\Users\avira\Desktop\first.py (3.8.2)

File Edit Format Run Options Window Help

```
stu_info = [10,'abcd', 25, 'MCA', 4534567877, 'xyz']
stu_info.insert(4, "final")
print(stu_info)
```

Output-:

```
[10, 'abcd', 25, 'MCA', 'final', 4534567877, 'xyz']
>>>
```

6. pop( )-: it will remove the last element from the list. It can also remove the element based on the given index.

Ex).

```
stu_info = [10,'abcd', 25, 'MCA', 4534567877, 'xyz']
stu_info.pop()
print(stu_info)
stu_info.pop(2) #pop can take index as argument
print(stu_info)
```

Output-:

```
[10, 'abcd', 25, 'MCA', 4534567877]
[10, 'abcd', 'MCA', 4534567877]
>>>
```

Q). program to multiply the list. (important)

Sol).

#doubling of the elements is possible.

```
stu_info = [10,'abcd', 25, 'MCA', 4534567877, 'xyz']
s1 = stu_info*2
print(s1)
```

Output-:

```
[10, 'abcd', 25, 'MCA', 4534567877, 'xyz', 10, 'abcd', 25, 'MCA', 453456
7877, 'xyz']
>>>
```

fundamental data types-: int, float, bool, complex.

Q). take a list and show some examples of indexing in list?

Sol).

```
li = [10, 20, 15, 'AVANISH', 'MCA']
print("values on 2 index:",li[2])
print("values on -2 index:",li[-2])
print("values on 5th index inside 3rd index :",li[3][5])
```

Output-:

```
values on 2 index: 15
values on -2 index: AVANISH
values on 5th index inside 3rd index : S
>>>
```

Q). describe the tuple data type in python. (important)

Sol). tuple is also a type that contains heterogeneous type of elements but tuple is immutable type of data type.

- it is randomly collection of elements.
- to create tuple one should use ( ) or tuple( ).
- tuple is immutable. i.e. tuple is read only collection of elements. which means we cannot update a tuple.
- so, we use tuple where we already know that we will never change the values.
- iterations are faster in tuple.

Ex-:

```
em_tpl = tuple() # you can create an empty tuple
print(em_tpl, type(em_tpl))
tpl = (10,'abcd', 25, 'MCA', 4534567877, 'xyz')
print(tpl)
```

Output-:

```
() <class 'tuple'>
(10, 'abcd', 25, 'MCA', 4534567877, 'xyz')
>>>
```

we can define list inside tuple.it is perfectly valid.

Ex).

stu_info=(10, 20, "name", [30, 35], 20.5)

Q). write a program that show we cannot update into tuple data type. Which is its immutability concept.

Sol). see if we tried to update the tuple, we get the error

```
stu_info = (10,'abcd', 25, 'MCA', 4534567877, 'xyz')
stu_info.append(999)
print(stu_info)
stu_info[0]=20
print(stu_info)
```

Output-:

```
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\first.py", line 3, in <module>
    stu_info.append(999)
AttributeError: 'tuple' object has no attribute 'append'
>>>
```

Note-: we can extract element on the basis of their index position.

Ex-:

```
stu_info = (10,'abcd', 25, 'MCA', 4534567877, 'xyz')
print(stu_info[4])
```

```
4534567877
>>>
```

Q). program to multiply the tuple.

Sol).

```
stu_info = (10,'abcd', 25, 'MCA', 4534567877, 'xyz')
s1 = stu_info*2
print(s1)
```

#here I'm not changing the tuple. I'm just creating a new tuple.

dict

Q). describe **dict data type** in python?

Sol).

- dictionary where the elements are stored in the pair/form of keys and values.
- you must have seen dictionary their elements are stored in the form of words and meanings.

- dict key values are represented inside curly braces {}. it is mutable.

Note-: list, tuple, set, frozenset, range, bytes, bytearry all these data types are applicable to represent individual objects. but if my requirement is to represent group of objects.

- or key-value pair (like-: id:12, name: avi, adds:prj)
- key cannot be duplicate but values are allowed to be duplicate. heterogeneous key and values are allowed.
- Dictionary is also known as mapping in python.

Q). program to create empty dictionary and store elements into that dictionary.

Sol).

```
s = {}    # empty dictionary
#s = dict() #empty dictionary.
print(s, type(s))
s['id']=10  # inserting a key as id and value as 10
s['name']='ravi'  # inserting a key as id and value as 10
s['adds']='delhi'
s[20]= 123
s[20]='rahul'
print(s)
```

Output-:

```
{} <class 'dict'>
{'id': 10, 'name': 'ravi', 'adds': 'delhi', 20: 'rahul'}
>>>
```

Q). explain methods in dict data type ?

Sol).

1. get( )-: it returns the value of the specified key. If you are searching a key which is not present in your dictionary then it will return you None.

Ex).

```
s = {'id':10, 'Name':'avi', 'Mob': 65767655676, 'Adds':'prj','Age':35}
k= s.get('Name')
print(k)
k= s.get('pin')
print(k)
```

Output-:

```
{} <class 'dict'>
{'id': 10, 'name': 'ravi', 'adds': 'delhi', 20: 'rahul'}
>>>
```

2. items( )-: it returns a list containing a tuple for each key value pair.

Ex).

```
s = {'id':10, 'Name':'avi', 'Mob': 65767655676, 'Adds':'prj', 34:'rra'}
print(s.items())
```

Output-:

```
dict_items([('id', 10), ('Name', 'avi'), ('Mob', 65767655676), ('Adds',
'prj'), (34, 'rra')])
>>>
```

3. keys( )-: it returns a list containing the dictionary's keys. (important)

Ex).

```
s = {'id':10, 'Name':'avi', 'Mob': 65767655676, 'Adds':'prj', 34:'rra'}
print(s.keys())
```

Output-:

```
dict_keys(['id', 'Name', 'Mob', 'Adds', 34])
>>>
```

4. update( )-: it updates the dictionary with the specified key-value pairs.

Ex). updating id and name key and add one more into given dictionary-:

```
di = {'id':10, 'Name':'avi', 'Mob': 65767655676, 'Adds':'prj', 34:'rra'}
d1 = {'id':12, 'Name':'tarun', 'age':20}
di.update(d1)
print(di)
```

Output-:

```
{'id': 12, 'Name': 'tarun', 'Mob': 65767655676, 'Adds': 'prj', 34: 'rra'
, 'age': 20}
>>>
```

11. values( )-: it returns a list of all the values in the dictionary. (important)

Ex).

```
s = {'id':10, 'Name':'avi', 'Mob': 65767655676, 'Adds':'prj', 34:'rra'}
print(s.values())
```

output-:

```
dict_values([10, 'avi', 65767655676, 'prj', 'rra'])
>>>
```

Q). describe the **set data type** in python?

Sol).

- we use set when we don't want duplicates as well as we don't want preserved order.
- set is defined by values separated by comma inside curly braces { }. Items in a set are not ordered.
- a = {11,20,9,7,4}
- set is mutable data type.
- Sets have unique values. They eliminate duplicates.
- We can perform set operations like union, intersection on two sets.
- indexing and slicing is not acceptable in set. because order is not there.
- set is growable.

Q). write a program to create a empty set also store some data?

Sol).

```
s = set()
print(s,type(s))
s = {10,20,30,50,16,9}
print(s)
n = set([0, 1, 2, 3, 4, 5])    #using set method
print(n)
```

Output-:

```
set() <class 'set'>
{9, 10, 16, 50, 20, 30}
{0, 1, 2, 3, 4, 5}
>>>
```

Q). write a program in set that shows we cannot store duplicate values in set.

Sol).

```
s = {10,20,30,50,16,9, 20,9}
print(s)
```

Output-:

```
{9, 10, 16, 50, 20, 30}
>>>
```

Q). write a program in set that shows set is mutable. And use of remove () method in set

Sol)

```
s = {10,20,30,50,16,9}
s.add('avi')
print(s)
s.remove(9)
print(s)
```

Output-:

```
{9, 10, 16, 50, 20, 'avi', 30}
{10, 16, 50, 20, 'avi', 30}
>>>
```

Q). write a program that find out union and intersection set and difference set?

Sol).

```
s1 = {'tarun', 20,11, 70.65, 'prj'}
s2 = {'o_level', 22, 12, 55.9, 7, 11, 'prj'}
s3 = s1.union(s2)
s4 = s1.intersection(s2)
print(" union element:\n", s3)
print('intersection elements\n:',s4)
s5 = s1.difference(s2)
print('difference set:\n',s5)
```

Output-:

```
 union element:
 {70.65, 7, 11, 12, 'prj', 20, 55.9, 'tarun', 22, 'o_level'}
intersection elements:
 {11, 'prj'}
difference set:
 {'tarun', 20, 70.65}
>>>
```

Disjoint set-: any set is called disjoint set if there is no common element between two compared sets.is returns True or False

Subset-: a set(a) is called subset if all the elements from that set(a) is present into another set(b).

Q). write a program in set to find out disjoint, update and check weather a set is subset or not?

Sol).

```
a = {1,2,3}
b = {1,2,3,4,5}
d = a.isdisjoint(b)
print('disjoint set: ',d)
c = a.issubset(b)
print(c)
a.update(b)
print(a)
```

Output-:

```
disjoint set:  False
True
{1, 2, 3, 4, 5}
>>>
```

Q). describe slice operator in python? (important)

Sol).

- To provide the flexibility to the programmers, python provides -ve (negative) indexing concept.
- It is most commonly used operator.
- Just like cutting the cake into pieces.
- create substrings of the string we use string slicing.
- there are multiple places where we can use slice operator. like (list, tuple, str, dict, etc)
- s[begin : end]
- it returns substring from begin end-1 index.
- if i search s[-1:-4], it will return nothing because begin value should be lower and end value should be higher.
- to fetch the result, here you can either provide begin or end data.
- s[:end]
- s[begin :]

- Syntax = > s[begin : end : step]

Q). show some examples how we can do slicing in list and strings? (important)

Ex-:

A). suppose we have a list and we want to extract the elements of that list from 3rd to 6th index.

Sol).

```
li = [10,20,30,35,76,99,87,43]
print("values from 3rd index to 6th index position:",li[3:7])
print("values from start index to 6th index position:",li[:6])
```

Output-:

```
values from 3rd index to 6th index position: [35, 76, 99, 87]
values from start index to 6th index position: [10, 20, 30, 35, 76, 99]
>>>
```

B). suppose if we have a string and we want to extract elements on the base of their index number-: (important)

Sol).

```
s = "AVANISH SINGH"
print("elements starts with 1st index and stops at 3rd index=",s[1:4])
print("elements starts with -1 index and stops at -4 index=",s[-1:-4])
#no output, because given begin index must always be
#smaller then end index. or give the step as -ve.
print("elements starts with -1 index and stops at -4 index and step is -1="
      ,s[-1:-4:-1])
print("elements starts with -4 index and stops at -1 index=",s[-4:-1])
print("elements starts with 1 index and stop is not specified=",s[1:])
print("elements starts with -1 index and stop is not specified=",s[-1:])
print("elements starts is not specified but stops at 4 index=",s[:4])
print("elements starts and stop both index not specified =",s[:])
print("elements starts with 1 index and stops at 50 index=",s[1:50])

print("elements starts with 0 index and stops at 7 index and step is 2=",
      s[0:7:2])
print("elements start and stop index is not specified but step is 3=",
      s[::3])
print("elements start and stop index also step is not specified",s[::])
print("elements start and stop index is'nt specified but step is -2=",s[::-2])
```

Output-:

```
elements starts with 1st index and stops at 3rd index= VAN
elements starts with -1 index and stops at -4 index=
elements starts with -1 index and stops at -4 index and step is -1= HGN
elements starts with -4 index and stops at -1 index= ING
elements starts with 1 index and stop is not specified= VANISH SINGH
elements starts with -1 index and stop is not specified= H
elements starts is not specified but stops at 4 index= AVAN
elements starts and stop both index not specified = AVANISH SINGH
elements starts with 1 index and stops at 50 index= VANISH SINGH
elements starts with 0 index and stops at 7 index and step is 2= AAIH
elements start and stop index is not specified but step is 3= ANHIH
elements start and stop index also step is not specified AVANISH SINGH
elements start and stop index is'nt specified but step is -2= HNSHIAA
>>>
```

Q). write a program to reverse a string. (important)

Sol).

```
s = 'tarun chaudhary'
print(s[::-1])
```

Output-:

```
yrahduahc nurat
>>> 
```

Q). define **Operators and their types** in python? (important)

Sol). the one who's performing some operations. (+, -, etc)

Ex-: 10 + 20 = 30

Types-:

1. Arithmetic operators
2. Relational operator or comparison operator.
3. Logical operator
4. Bitwise operator
5. Assignment operator
6. Special operator

Q). define Arithmetic operator in python?

Sol).

- addition operator (+) = it is also applicable for string operator. it is also string concatenation operator. (both values should be strings)
- subtraction operator (-)
- multiplication operator (*)
- division operator (/) = result of division operator is always a float value. you never get int value.
- modulo operator (%) = it returns the reminder value.
- floor division operator (//) = it return the floor value which can be either float or int type.(important)
- exponent operator (**) = returns power of the given values.

Q). simple program that uses arithmetic operators

Sol).

```
a = 12
b = 5
print("Addition of a and b is:",a+b)
print("substraction of a and b is:",a-b)
print("Multiplication of a and b is:",a*b)
print("division of a and b is:",a/b)
print("Reminder of a and b is:",a%b)
print("floor division of a and b is:",a//b)
print("Power of a and b is:",a**b)
```

Output-:

```
Addition of a and b is: 17
substraction of a and b is: 7
Multiplication of a and b is: 60
division of a and b is: 2.4
Reminder of a and b is: 2
floor division of a and b is: 2
Power of a and b is: 248832
>>>
```

Q). define Relational operators (>, >=, <, <=) in python?

Sol).

These operators are for checking the conditions and return the float result. they return the Boolean result (either True or False).

- it is also applicable for string values. it returns the results based on alphabetical order.
- chaining of relational operator is also possible.

Q). write a program that take two inputs and check relational operators working?

Sol).

```
a = 20
b = 13
print("a>b:", a>b)
print("a<b: ", a<b)
print("a>=b: ", a>=b)
print("a<=b: ", a<=b)
m = 'smaller'
n = 'larger'
print("m>n:", a>b)
print("m<n: ", a<b)
print("m>=n: ", a>=b)
print("m<=n: ", a<=b)
```

Output-:

```
a>b: True
a<b:  False
a>=b:  True
a<=b:  False
m>n: True
m<n:  False
m>=n:  True
m<=n:  False
>>>
```

Q). program that checks the condition for chaining of relational operators?

Sol).

```
a, b, c, d, e = 10, 20, 30, 40, 50
print(a<b<c<d<e)
print(a>b>c>d>e)
print(a<b<c<d>e)
```

Output-:

```
True
False
False
>>>
```

Q). define comparison operator (==). (important)

Sol).

- equality operator is there to compare the contents. we are not going to get any errors. It returns the Boolean result (either True or False).

- if content is same return true if not same returns false.
- chaining concept is also applicable.

Q). define Logical Operator? (important)

Sol).

- these operators you can apply for all types. they return the Boolean result (either True or False).
- and (&&) = only if all the arguments are true then it returns true. (True and False = False)
- or (||) = if any one of the arguments is true then it returns true. (True or False = True)
- not (!) = if argument is true returns false and vice versa. (not True = False)

Q). define Bitwise operators?

Sol).

- Bitwise operators are used to compare (binary) numbers:
- for operating the bits.
- these operators applicable only for int and bool type of data.

1. & (and) -: if both bits are 1 then returns 1 otherwise 0.

```
1 0 1 0
1 1 1 1
---------
1 0 1 0
---------
```

2. | (or)-: if at least one bit is 1 then returns 1 otherwise 0.

```
1 0 1 0
1 1 1 1
---------
1 1 1 1
---------
```

3. ^ (x-or)-: exclusive or, i.e. if both bits are different then returns 1 otherwise 0.

1 0 1 0

1 1 1 1

---

0 1 0 1

---

4. ~ (compliment operator)-: transposing 1 and 0. i.e. 1 become 0 and 0 become 1.

Q). describe Python Assignment Operators?

Sol). Assignment operators are used to assign values to variables.

Operator	Example	Treat as
=	a = 20	a = 20
+=	a += 5	a = a + 5
-=	a -= 3	a = a - 3
*=	a *= 5	a = a * 5
/=	a /= 15	a = a / 15
%=	a %= 27	a = a% 27
//=	a//= 27	a = a // 27
**=	a **= 9	a = a ** 9
&=	a &= 7	a = a & 7
|=	a |= 9	a = a | 9
^=	a ^= 15	a = a ^ 15
>>=	a >>= 10	a = a >> 10
<<=	a <<= 10	a = a << 10

Q). describes python Special operators? (important)

Sol). There is some special type of operators like-:

1. Identity operators

2. Membership operators

- Identity operators is used for address comparison.
- is operator is used here to check weather both values have same id or not.
- is not operator will return true if both values are not same.

Membership operators-:

- Membership operators are used to test if a sequence is presented in an object
- in -: Returns True if a sequence with the specified value is present in the object
- not in -: Returns True if a sequence with the specified value is not present in the object

Q). example of identity operator and mutability-:

```
x = ["apple", "banana"]   #1234455445

y = ["apple", "banana"]

z = x   #

print(id(x))

print(id(y))

print(x is y)

print(x is not y)

# returns True because z is the same object as x

print(x is z)


x = ("apple", "banana")   #1234455445

y = ("apple", "banana")

z = x   #

print(id(x))
```

```
print(id(y))

print(x is y)

print(x is not y)

# returns True because z is the same object as x

print(x is z)


li = ['tarun', 1, 78]

print(id(li))

li[0]='chaudhary'

print(li, id(li))
```

Q). program to check weather list has the same elements as list x. (important)

Sol).

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z)
# returns True because z is the same object as x
print(x is y)
```

Output-

```
True
False
>>>
```

Q) program to check whether any specified element is in the given list or not using membership operator. (in/not in)

Sol).

```
c1 = ['red', 'blue', 'black']
print('blue' in c1)
print('yellow' not in c1)
```

Output-:

```
True
True
>>>
```

Q). describe end attribute of print method with example? (important)

Sol).

- if you want to add something to the end of the element and don't want to break the line.
- default value of end is new line characters.

Ex).

```
print("watching", end='-')
print("The Movie", end='-')
print(300)
```

output-:

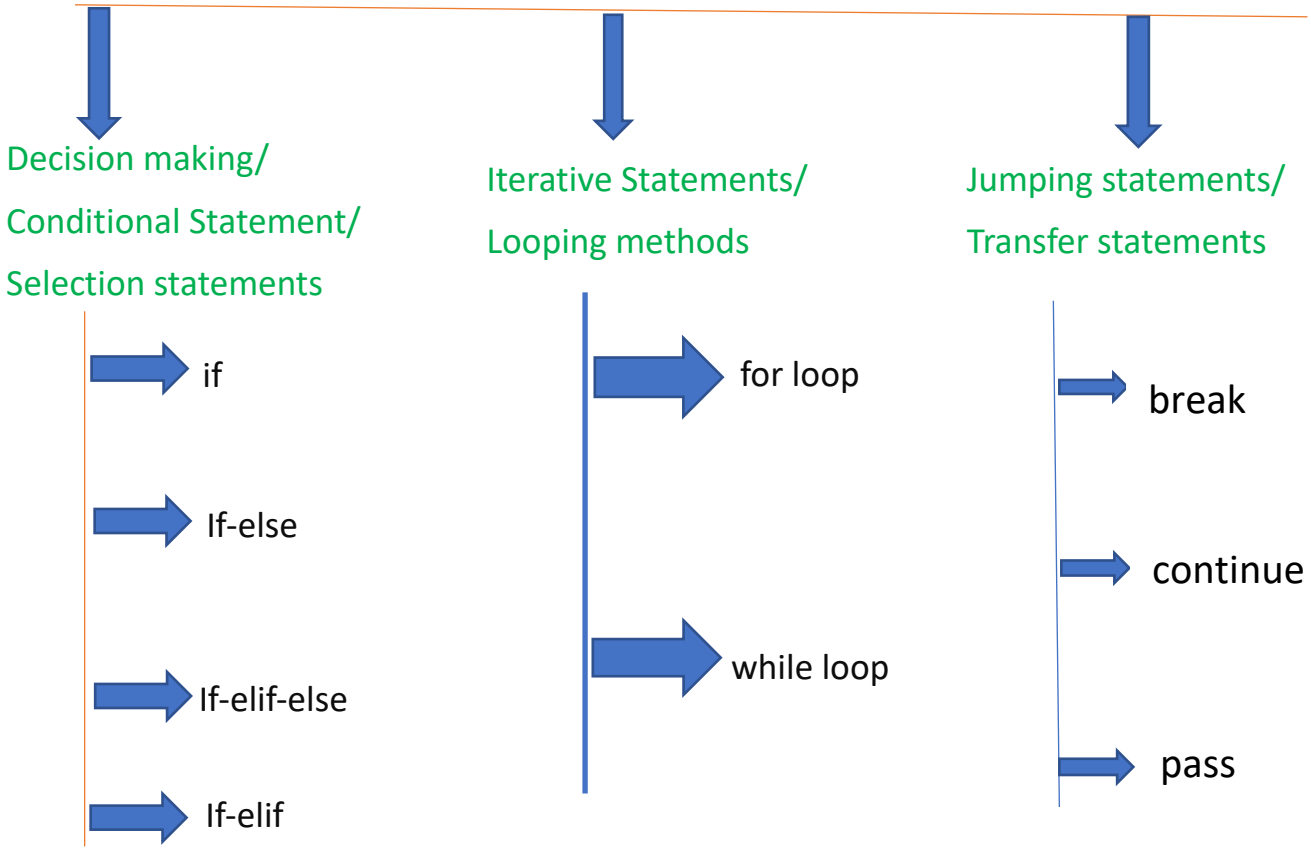
```
watching-The Movie-300
>>>
```

Unit - 4

Control Flow

in which order the statements are going to be executed at run time.

Types of Control flow statements



Q). what is decision making/ conditional/ selection statements.

Sol).

- In python when your statement or body of the program is going to decide on the base of some condition, then those statements are called as conditional statement.
- These conditions return a bool result. (either True or false).
- in python you should compulsorily use colon with if, else and elif.
- else is always optional.
- switch is not available in python.

Q). write down the types of conditional statements with syntax? (important)

Sol).

1. if
2. if-else
3. if-elif-else
4. if-elif

1. if

```
if condition:
    body of true condition.
```

2. if – else

```
if condition:
    statement of true condition
else:
    statement of false condition
```

3. if-elif-else

```
if condition1:
    statement for condition1 true
elif condition2:
    statement for condition2 true
```

elif condtion3:

statement for condition3

.....

.....

elif condition N-1:

expression N-1

else:

statement for if all the above conditions false

4. if-elif

syntax-: if- elif

if condition1:

expression1

elif condition:

expression2

elif condition3:

expression3

Q). write a program to check which is bigger number among two? (important)

Sol).

```
n1=int(input("enter the first number:"))
n2=int(input("Enter the second number:"))
if n1>n2:
    print(n1,"is bigger then",n2)
else:
    print(n2,"is bigger")
```

Output-:

```
enter the first number:30
Enter the second number:27
30 is bigger then 27
>>>
```

Q). write a program to check weather user have input the correct credentials for login or not?

Sol).

```
u_name = input("enter the user name: ")
pasw = input("enter your password:")
if u_name=='python' and pasw=='12345':
    print("you have successfully logged in!!")
else:
    print("check your user name and password again!")
```

Output-:

```
enter the user name: python
enter your password:12345
you have successfully logged in!!
>>>
```

Q). write a program that checks weather the entered mobile number and password length is correct or not?

Sol).

```
name = input("Enter Your name: ")
mobile = input("Enter your mobile number: ")
aadhar = input("Enter Your Aadhar number: ")
if len(mobile) == 10 and len(aadhar) == 12:
    print("your information is correct Mr.", name)
else:
    print("Your mobile number or Aadhar number isn't accurate")
```

Output-:

```
Enter Your name: avanish
Enter your mobile number: 5643655489
Enter Your Aadhar number: 345345422
Your mobile number or Aadhar number isn't accurate
>>>
```

Q). write a program to print the grade of a student in python?

Sol).

```
name = input("Enter Your name:")
num = int(input("Enter the number you get in python:"))
if num>=90 and num<=100:
    print(name, "got 'A+' Grade")
elif num>=80 and num<90:
    print(name, "got 'A' Grade")
elif num>=70 and num<80:
    print(name, "got 'B+' Grade")

elif num>=60 and num<70:
    print(name, "got 'B' Grade")

elif num>=50 and num<60:
    print(name, "got 'C' Grade")
else:
    print(name, "You are Fail in this subject")
```

Output-:

```
Enter Your name:Deepak
Enter the number you get in python:93
Deepak got 'A+' Grade
>>>
```

Q). Write a program that take input in year and return weather the year is leap year or not? (important)

Sol).

```
year = int(input("Enter year: "))
if year%4:
    print("not leap year")
elif year%100==0 and year%400!=0:
    print("not leap year")
else:
    print("it's leap year")
```

Output-:

```
Enter year: 1997
not leap year
>>>
```

Q). Write a program that take input as height(meters) and weight(kg), returns Body mass Index (BMI) with appropriate statement.

Sol).

```
height = eval(input("Enter Your height:"))
weight = eval(input("Enter Your weight:"))
bmi = weight/height**2
print(bmi)
if bmi<15:
    print("very severely underweighted")
elif bmi<16:
    print("severely underweighted")
elif bmi<18.5:
    print("underweighted")
elif bmi<25:
    print("noraml(healthy weighted)")
elif bmi<30:
    print("overweighted")
elif bmi<35:
    print("obese class 1(moderately obese)")
elif bmi<40:
    print("obese class 2(severely obese)")
else:
    print("obese class 3(very severely obese)")
```

Output-:

```
Enter Your height:1.70
Enter Your weight:63
21.799307958477511
noraml(healthy weighted)
>>>
```

Q). Write a program of nested if that calculate the highest number among the Three numbers.

Sol).

```
num1 = eval(input("1st number:"))
num2 = eval(input("2nd number:"))
num3 = eval(input("3rd number:"))
if num1>num2:
    if num1>num3:
        highest = num1
    else:
        highest = num3
else:
    if num2>num3:
        highest = num2
    else:
        highest = num3
print("highest number is :", highest)
```

Output-:

```
1st number:20
2nd number:23
3rd number:18
highest number is : 23
>>>
```

we can use max method of python to find maximum number given as below-:

```
num1 = int(input("enter first number:"))
num2 = int(input("enter second number:"))
num3 = int(input("enter third number:"))
print("maximum number is:",max(num1,num2,num3))
print("minimum number is:",min(num1,num2,num3))
```

Output-:

```
enter first number:65
enter second number:30
enter third number:55
maximum number is: 65
minimum number is: 30
>>>
```

Q). Write a program that take input in celcius or fahrenheit and return the result after converting it to its appropriate type.

Sol).

```
temp = input("enter the temprature you want to convert(e.g. 45f,102c)")
degree = int(temp[: -1])
i_convention = temp[-1]
if i_convention.upper()=="C":
    result = int(round((9*degree)/5+32))  # celsius to farenheit
    o_convention = "Fahrenheit"
elif i_convention.upper()=="F":
    result = int(round((degree-32)*5/9))   # farenheit to celsius
    o_convention = "Celsius"
else:
    print("Enter appropriate convention!")
    quit()
print("the temprature in",o_convention,"is",result,"degree")
```

Output-:

```
enter the temprature you want to convert(e.g. 45f,102c)55c
the temprature in Fahrenheit is 131 degree
>>>
```

LOOPS

Q). describe iterative statements in python And loops? (important)

Sol). iterative statements mean that when we require to repeat our tasks again and again, in these cases we use loops for iterative statements.

We use loops to iterate the conditions. When we required to repeat the task again and again.

Loops

Where the group of statements are going to execute iteratively.

Q). Write down the types of loops in python?

Sol). there are two types of iterative statements present in python-:

1. for Loop
2. while Loop

For Loop

Q). describe for loop with syntax? (important)

Sol).

- If you are going to perform some actions for each element in the sequence, then use for loop.
- for loops are used for sequential traversal.
- Using for loop we can iterate every element inside a given sequence. (like-: list, str, tuple, dict, set, range, etc)

- **Syntax**

  for each_element in sequence:

  action/expressions/body

Note-:

- this body will be executed for every element present in the sequence. This sequence can be of any data type (like- str, range, list, tuple, dict, etc).
- Increment and decrement operators are not available in python.

Q). describe else statement with syntax in for loop?

Sol).

- We can also combine else statement with for loop. Here else block will run only after the loop complete its execution.
- If you are terminating for loop forcefully then else block will not execute
- The else keyword in a for loop specifies a block of code to be executed when the loop is finished.

Syntax

for variable in sequence:

action/expressions/body

Else:

action/expressions/body

Note-: All the data types except first four, you have encountered so far that all are collection or container type of data type.

- These include the string, list, tuple, dict, set, and frozenset types.

Q). write a program that prints each sequence of given string one by one.
Sol).

```
name = "BAHUBALI"
for x in name:
    print(x)
```

Output-:

```
B
A
H
U
B
A
L
I
>>>
```

Q). write a program that count the number of characters given in the string?
Sol).

```
name = "BAHUBALI"
count = 0
for x in name:
    count +=1
    print(x)
else:
    print("no. of character=",count)
```

Output-:

```
B
A
H
U
B
A
L
I
no. of character= 8
>>>
```

Q). What is range function in python? (important)

Sol).

- to represent the range of values.
- it is a data type as well as function.
- it represents sequence of values.
- suppose you want to print values from range of 0 to 10. then use range (11).
- (0 to end-1)

Note-: there are three ways how you can use range function

- range (stop) # here starting range is by default 0.
- range (start, stop)
- range(start, stop, increment)

Ex-: suppose you have written like-: range (1, 12). It means your range function contains the numbers like-: 1,2,3,4,5,6,7,8,9,10,11

Q).print all numbers between 0 to 9.

Sol).

```
for i in range(0,10):
    print(i)
```

Output-:

```
0
1
2
3
4
5
6
7
8
9
>>>
```

Q). write a program to print all even numbers inside a given list?

Sol).

```
li = [2, 7, 5, 3, 10, 55, 60, 32]
print(li)
print("even numbers:")
for i in li:
    if i%2==0:
        print(i)
```

Output-:

```
[2, 7, 5, 3, 10, 55, 60, 32]
even numbers:
2
10
60
32
>>>
```

Q). write a program to print some of all elements/sequences inside a list?

Sol).

```
li = [10,20,30,60,77,54,99,23]
som = 0
for i in li:
    som = som+i
print("addition of the numbers in the list is=",som)
```

Output-:

```
addition of the numbers in the list is= 373
>>>
```

Q). write a program to print 10 numbers in reverse order? (important)

Sol).

```
for i in range(10,0,-1):
    print(i)
else:
    print("We have reached to the last element of the given range")
```

Output-:

```
10
9
8
7
6
5
4
3
2
1
We have reached to the last element of the given range
>>>
```

Q). write a program to print each element of dictionary in form of key and values? (important)

Sol).

```
di = {'id': 10, 'name': 'avi', 'age': 25}
for k, v in di.items():
    print('key =', k, ',', ' values =', v)
```

Output-:

```
key = id ,  values = 10
key = name ,  values = avi
key = age ,  values = 25
>>>
```

Q). write a program to print the table of any number?

Sol).

```
num = 5
for i in range(1,11):
    print(num, 'x', i, '=', num*i)
```

Output-:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
>>>
```

Q). describe nested for loop with syntax in python?

Sol).

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop“

Syntax

for variable in sequence:

for variable in sequence:

action/expressions/body

action/expressions/body

Q). write a program to print the table of numbers inside given range?

Sol).

```
for i in range(5,8):
    for j in range(1,11):
        print('{0} * {1} = {2}'.format(i,j,i*j))
    print('================================')
```

output-:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
==============================
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
6 * 5 = 30
6 * 6 = 36
6 * 7 = 42
6 * 8 = 48
6 * 9 = 54
6 * 10 = 60
==============================
```

Q). write a program to print the factorial result of given number?

Sol).

```
num = int(input("Enter Any number: "))
f=1
som=0
for i in range(1,num+1):
    f = f*i
som = som+f
print("factorial result is=", som)
```

Output-:

```
Enter Any number: 6
factorial result is= 720
>>>
```

Q). Write a program to make the name of a person in short form.

Sol).

```
name = input("Enter your name: ").split(' ')
res = str()
length = len(name)-1
for i in range(length):
    res = res + name[i][0] + "."
res = res + name[-1]
print("You name in short form is-: ", res)
```

output-:

```
Enter your name: avanish singh rajwaar
You name in short form is-:  a.s.rajwaar
>>>
```

Q). Write a program to print series of prime numbers between the range?

Sol)

```
n1 = int(input("Enter the prime number from: "))
n2 = int(input("Enter the prime number upto: "))
for num in range(n1, n2+1):
    for i in range(2, num):
        if(num % i == 0):break
    else:
        print(num)
```

output-:

```
Enter the prime number from: 50
Enter the prime number upto: 100
53
59
61
67
71
73
79
83
89
97
>>>
```

Q). Write a program to print all factors of any given number?

Sol).

```
num = int(input("enter any number : "))
for i in range(1, num+1):
    if(num % i == 0):
        print(i)
```

Output-:

```
enter any number : 80
1
2
4
5
8
10
16
20
40
80
>>>
```

Q) write a program to print Fibonacci series for any number of terms?

Sol).

```
term =int(input("Enter the term up to the series you want ? "))
first = 0
second = 1
for i in range(term):
    print(first)
    temp = first
    first = second
    second = temp + second
```

Output-:

```
Enter the term up to the series you want ? 11
0
1
1
2
3
5
8
13
21
34
55
>>>
```

Q). write a program to search weather a name of employee exists inside a company or not, if exist also print his employee id number?

Sol).

```
search_name = input("Enter the name of employee: ")
ids = {1: 'avanish', 2: 'ravi', 3: 'xavi'}
for emp_id,emp_name in ids.items():
    if emp_name == search_name:
        print(search_name,"is present in our data his id is:",emp_id)
        break
else:
    print('No entry with that name found.')
```

Output-:

```
Enter the name of employee: ravi
ravi is present in our data his id is: 2
>>>
```

Q). describe enumerate function in python with syntax? (important)

Sol).

enumerate () method adds a counter to an sequence and returns it in a form of enumerate object. This enumerate object can then be used directly in for loops or be converted into a list of tuples using list() method.

syntax -:

for index, item in enumerate ([list of elements]):

Q). example of enumerate function to print elements of list.

Sol).

```
li = [1,2,3,4,5,6,1,2,3,4,5]
for index, item in enumerate(li):
    print(index+1, '::', item)
```

Output-:

```
1 :: 1
2 :: 2
3 :: 3
4 :: 4
5 :: 5
6 :: 6
7 :: 1
8 :: 2
9 :: 3
10 :: 4
11 :: 5
>>>
```

Q). write a python program to interchange first and last element of a list.

Sol).

```
li = [eval(x) for x in input("Enter the elements of a list:").split(",")]
size = len(li)
temp = li[0]
li[0] = li[size-1]
li[size-1] = temp
print(li)
```

output-:

```
Enter the elements of a list:56,897,24,76,765,167
[167, 897, 24, 76, 765, 56]
>>>
```

Q). write a program to test whether a number is Armstrong or not

Sol).

```
num = input("Enter any number-: ")
if(num.isdigit()):
    length = len(num)
    res = 0
    for digit in num:
        res = res + (int(digit)**length)
    else:
        if(res == int(num)):
            print(num, "is an armstrong number")
        else:
            print(num, "is not an armstrong number")
else:
   print("invalid Input")
```

Output-:

```
Enter any number-: 370
370 is an armstrong number
>>>
```

Q). write a program to print the given pattern?

Sol).

```
Enter the number of rows-: 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
>>>
```

```
num = int(input("Enter the number of rows-: "))
for i in range(1, num+1):    #5
    for j in range(1, i+1):  #range(1,1)
        print(j, end=' ')
    print()
```

Q). write a program to print the given pattern?

Sol).

```
Enter the number of rows-: 6
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
>>>
```

```
num = int(input("Enter the number of rows-: "))
for i in range(1, num+1):
    for j in range(1, i+1):
        print(i, end=' ')
    print()
```

Q). write a program to print the given pattern?

Sol).

```
Enter the number of rows-: 7
*
**
* *
*  *
*   *
*    *
*******
>>>
```

```
num = int(input("Enter the number of rows-: "))
for i in range(num):
    for j in range(num):
        if(j == 0 or i == num-1 or i == j):
            print('*', end='')
        else:
            print(end = ' ')
    print()
```

Q). write a program to print the given pattern?

Sol).

```
* * * * *
 *   *
  * *
   * *
    *
```

```
num = int(input("Enter the number of rows-: "))
for row in range(1, num+1):
    for col in range(1, row+1):
        if(row == 0 or col == num-1 or row == col):
            print('*', end='')
        else:
            print(end = ' ')
    print()
```

Q). write a program to print floids triangle pattern?

Sol).

```
Enter the number of rows-: 5
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
>>>
```

```
num = int(input("Enter the number of rows-: "))
n = 1
for row in range(1, num+1):
    for column in range(1, row+1):
        print(n, end=' ')
        n +=1
    print()
```

Q). print the given pattern?

Sol).

```
Enter your string-: python
p
p y
p y t
p y t h
p y t h o
p y t h o n
>>>
```

```
name = input("Enter your string-: ")
length = len(name)
for row in range(length):
    for col in range(row+1):
        print(name[col], end = ' ')
    print()
```

While Loop

Q). describe While Loop with syntax? (important)

Sol).

- when we already know that how many times i must do the iterations, then we go for "for loop".
- if i don't know iterations in advance (i.e., how many times i must iterate), then go for "while loop".
- While Loops is used to execute a block of statements repeatedly until a given condition is satisfied.

syntax:

```
while condition:
        body/expressions
```

- when condition returns False our while will stop. you can also use else with while loop.

Syntax-:

```
while condition:
        body
else:
    body
```

Q). define infinite loop in python with example?

Sol).

a loop which has the capability to iterate infinite times.

Example-:

```
i = 0
while True:
    i = i+1
    print(i)
```

Q). Write a program to print all odd numbers between 1 to 100.

Sol).

```
x = 1
while x <= 20:
    if(x % 2!= 0):
        print(x)
    x +=1
```

Output-:

```
1
3
5
7
9
11
13
15
17
19
>>>
```

Q). write a program to print Armstrong number between the ranges?

Sol).

```
start = int(input("Enter the starting range: "))
end = int(input("Enter the ending range: "))
for i in range(start, end+1):
    num = i
    res = 0
    length = len(str(i))
    while(i!=0):
        digit = i%10
        res += digit ** length
        i = i//10
    if(num == res):
        print(num)
```

Output-:

```
Enter the starting range: 50
Enter the ending range: 500
153
370
371
407
>>>
```

Q) write a program that shows while-else loop with example?

Sol).

when we are not forcefully stopping our loop

```
i = 0
while i < 4:
    i += 1
    print(i)
else: # Executed because no break statements are here
    print("else part has executed\n")
```

Output-:

when we are forcefully stopping our loop

```
i = 0
while i < 4:
    i += 1
    print(i)
    break
else: # Not executed as there is a break statement used.
    print("else part will not execute because of break statement")
```

Q). describe Jumping statements/Transfer statements in python? (important)

Sol). There are three jumping statements in python as shown below-:

1. break
2. continue
3. pass

break

- come out of something(loop).
- if we want to break our loop Execution forcefully, then we use break statement.
- we usually break our loop execution based on some conditions.

continue

As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

pass

- As the name suggests pass statement simply does nothing.
- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- It is like null operation, as nothing will happen is it is executed. Pass statement can also be used for writing empty loops. Pass is also used for empty control statement, function and classes.

Q). write a program that prints the table of numbers between 2 to 6. But it prints only 5 terms for each number in this range?

Sol).

```
for i in range(2,6):
    print('Table of ',i)
    for j in range(1,11):
        print(i*j)
        if (j == 5): break
```

```
Table of  2
2
4
6
8
10
Table of  3
3
6
9
12
15
Table of  4
4
8
12
16
20
Table of  5
5
10
15
```

Q). write a program that takes a string and print characters of that string, but if character found ‘i’ then break the printing of characters. (important)

Sol).

```
for val in "Avanish":
    if val == "i": break
    print(val)
print("The end")
```

Output-:

```
A
v
a
n
The end
>>>
```

Q). write a program that prints given string but it skips the character ‘i’ or ‘g’.

Sol).

#Use of continue statement inside the loop

```
for val in "string":
    if val == "i" or val == "g":
        continue
    print(val)
print("The end")
```

```
s
t
r
n
The end
>>> |
```

Ex). example of pass statement-: (important)

```
s = "python"
for i in s:
    if i == 'o':
        print('Pass executed')
        pass
    print(i)
```

Output-:

```
p
y
t
h
Pass executed
o
n
>>> |
```

Q). write a program to calculate the maximum and minimum element of given list. (important)

Sol).

```
li = [5, 8.9, 6, 9.8, 10, 3, 11.3]
print("maximum element is: ", max(li))
print("minimum element is: ", min(li))
```

Output-:

```
maximum element is:  11.3
minimum element is:  3
>>> |
```

Q). define deepcopy and shallow copy in python? (important)

Sol).

In **Shallow copy any changes** made to a copy of object **do reflect** in the original object.

Ex).

```
li1 = [10,22,76,65,89,54,89]
# i am going to do shallow copy
li2 = li1
#now i am making changes in the copied list
li2[4]=99
li2[3]=17
#again try to print li1 and li2
# you will the changes you have made on the copied list
# those change has been also applied on the orignal list is well
print("original list is:",li1)
print("copied list:",li2)
```

Output-:

```
original list is: [10, 22, 76, 17, 99, 54, 89]
copied list: [10, 22, 76, 17, 99, 54, 89]
>>>
```

deepcopy is a process in which a**ny changes** made to a copy of object **do not reflect** in the original object. In python, this is implemented using "**deepcopy()**" function.

Ex).

```
import copy
li1 = [10,22,76,65,89,54,89]
# i am going to do deep copy
li2 = copy.deepcopy(li1)
#now i am making changes in the copied list
li2[4]=99
li2[3]=17
#again try to print li1 and li2
# you will see the changes you have made on the copied list
# those change has not been applied on the orignal list.
print("original list is:",li1)
print("copied list:",li2)
```

Output-:

```
original list is: [10, 22, 76, 65, 89, 54, 89]
copied list: [10, 22, 76, 17, 99, 54, 89]
>>>
```

Q). Write a python program for counting the frequency of elements in a list using a dictionary.

```
list1 = ['a','b','a','c','d','c','c', 'd', 'f', 'a', 'b', 'c', 'd', 'e']
frequency = {}
for item in list1:
    frequency[item] = list1.count(item)
for key, value in frequency.items():
    print("% s -> % d" % (key, value))
```

Output-:

```
a ->  3
b ->  2
c ->  4
d ->  3
f ->  1
e ->  1
>>>
```

Q). write a program to print the frequency of each word in given sentence.?

Sol).

```
wordstring = 'it was the best of times it was the worst of times '
wordstring += 'it was the age of wisdom it was the age of foolishness'

wordlist = wordstring.split()
wordfreq = []
for w in wordlist:
    wordfreq.append(wordlist.count(w))
print("String\n" , wordstring +"\n")
print("List\n" , str(wordlist) + "\n")
print("Frequencies\n" + str(wordfreq) + "\n")
```

Output-:

```
String
 it was the best of times it was the worst of times it was the age of wisdom i
t was the age of foolishness

List
 ['it', 'was', 'the', 'best', 'of', 'times', 'it', 'was', 'the', 'worst', 'of'
, 'times', 'it', 'was', 'the', 'age', 'of', 'wisdom', 'it', 'was', 'the', 'age
', 'of', 'foolishness']

Frequencies
[4, 4, 4, 1, 4, 2, 4, 4, 4, 1, 4, 2, 4, 4, 4, 2, 4, 1, 4, 4, 4, 2, 4, 1]

>>>
```

Comprehension in python-:

Comprehensions in Python provide us with a short and concise way to construct new sequences (such as lists, set, dictionary etc.) using sequences which have been already defined. Python supports the following 4 types of comprehensions:

- List Comprehensions
- Dictionary Comprehensions
- Set Comprehensions
- Generator Comprehensions

List comprehension in python

List Comprehensions provide an elegant way to create new lists.

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Syntax-:

output_list = [output_exp for var in input_list if (var satisfies this condition)]

syntax-:

newlist = [*expression* for *item* in *sequence* if *condition* == True]

Note that list comprehension may or may not contain an if condition.

you can write multiple if statements. You can also write one or more than one for loops.

Ex-: use of range in list comprehension

```
newlist = [x for x in range(10)]
print(newlist)
newlist2 = [x for x in range(10) if x < 5]
print(newlist2)
```

Output-:

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

Ex-: print only those numbers from a list which is even.

Sol).

```
input_list = [1, 2, 3, 4, 4, 5, 6, 7, 7]

lc = [var for var in input_list if var % 2 == 0]

print("Output List using list comprehensions:", lc)
```

Output-:

```
Output List using list comprehensions: [2, 4, 4, 6]
>>>
```

Ex). show how you can use nested if conditions using list comprehension

Sol).

```
nlist = [i for i in range(1,21) if i%2==0 if i%3==0]
print(nlist)
```

Output-:

```
[6, 12, 18]
>>>
```

Ex-: suppose you want to print a list after excluding some elements.

Sol).

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if x != "apple"]
print(newlist)
```

Output-:

```
['banana', 'cherry', 'kiwi', 'mango']
>>>
```

List comprehension using if-else-:

Syntax-:

New_list = [expression_of_if if_condition else else_expression for item in sequence]

Ex-: take a list 20 numbers and check even odd. Print the number if it is even otherwise print odd.

Sol).

```
nlist = [i if i%2==0 else "odd" for i in range(1,21)]
print(nlist)
```

Output-:

```
['odd', 2, 'odd', 4, 'odd', 6, 'odd', 8, 'odd', 10, 'odd', 12, 'odd', 14, 'odd
', 16, 'odd', 18, 'odd', 20]
>>>
```

Dictionary Comprehensions:

we can also create a dictionary using dictionary comprehensions.

Syntax-:

output_dict = {key:value for (key, value) in iterable if (key, value satisfy this condition)}

Ex-: take a dictionary and make the cube of each value.

Sol).

```
input_list = [1,2,3,4,5,6,7]
dc = {var:var ** 3 for var in input_list if var % 2 != 0}
print("Output Dictionary using dictionary comprehensions:", dc)
```

Output-:

```
Output Dictionary using dictionary comprehensions: {1: 1, 3: 27, 5: 125, 7: 34
3}
>>>
```

Ex). take two lists and zip them together inside a dictionary?

Sol).

```
state = ['Gujarat', 'Maharashtra', 'Rajasthan']
capital = ['Gandhinagar', 'Mumbai', 'Jaipur']

dc = {key:value for (key, value) in zip(state, capital)}
print("Output Dictionary using dictionary comprehensions:",dc)
```

Output-:

```
Output Dictionary using dictionary comprehensions: {'Gujarat': 'Gandhinagar',
'Maharashtra': 'Mumbai', 'Rajasthan': 'Jaipur'}
>>>
```

function

Q). define function with syntax in python? (important)

Sol).

- A function is a set of statements that take inputs, do some specific computation and produces output. The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.
- Python provides built-in functions like print(), etc. but we can also create your own functions. These functions are called user-defined functions.
- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

syntax -:

def method_name(parameters):

"Documentation String" #optional

statements.....

.....

Ex-:

```
def hello(name):
    "this is a simple program"
    x = "Mr. "+name
    print(x)
hello("Avanish")
```

Output-:

```
Mr. Avanish
>>>
```

note-: function name is known as the identifier of the function. and parameter is an optional list of identifiers.

Note-: there are two types of functions -:

- Pre-defined/built-in/library function
  Functions which are already created and stored in python library are called built-in functions. For using these functions, we just need to call them with their names in our python file.
- User defined functions
  As its name suggests user defined functions are created by users. You can create your own function with any name and easily use in by calling it inside that python file.

Q). what are the types of parameters in function? (important)

Sol).

1. required
   required parameters are those parameters which are compulsory to provide by the programmer.

Ex-: def my_fun(name):

"here name is required parameter"

2. default

here default value will we passed in the argument for a variable.

Q). write a program that shows the use of default parameter in function?

Sol)

```
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)
# We call myFun() with only argument
myFun(10)
```

Output-:

```
x:  10
y:  50
>>>
```

3. keyword

here we specify argument name with values so that caller does not need to remember order of parameters.

Q). write a program that shows the use of keyword parameter/parameter in function of python? (important)

Sol) .

```
def my_function(child3, child2, child1):
    print("The youngest child is " , child3)
# calling function
my_function(child1 = "xyz", child2 = "abc", child3 = "mno")
```

Output-:

```
The youngest child is  mno
>>>
```

4. arbitrary

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

This way the function will receive a tuple of arguments

Q). write a program that shows the use of arbitrary parameter/keyword in function of python?

Sol)

```
def myFun(*argv):
    for arg in argv:
        print (arg)
myFun('Hello', 'Welcome', 'to', 'python Series', 'by', 'Er. Avanish Singh')
```

Output-:

```
Hello
Welcome
to
python Series
by
Er. Avanish Singh
>>>
```

Q). define calling process in function? (important)

Sol).

- At the end of the program when we call the function outside function body (calling function means writing the same name of the function which we have already declared and pass the actual value as an argument to that function.) and inside parenthesis we pass the value for required parameter variable. This process is called as calling the function process.
- Function always called outside the body of declared function.
- We can call a function one or more than one time.

Q). write a program to wish the good morning to your friend where you show the default parameter, documentation string and calling the function?

Sol).

```
def welcome(name):
    """
    This function welcomes to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
welcome('Python')    # calling the function
```

Output-:

```
Hello, Python. Good morning!
>>>
```

Q). define return statement in python with example? (important)

Sol).

A return statement is used to end the execution of the function call and "returns" the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.

Note: Return statement cannot be used outside the function.

Ex).

```
def absolute_value(num):
    """This function returns the absolute
             value of the entered number"""
    if num >= 0:
        return num
    else:
        return -num
print(absolute_value(7))              # function calling
print(absolute_value(-4))               # function calling
```

Output-:

```
7
4
>>>
```

Q). write a program of function that shows we can return multiple values together with return statement? (important)

Sol).

```
def calculator(a,b):
    som = a+b
    sub = a-b
    mul = a*b
    div = a/b
    fdiv = a//b
    power = a**b
    return som, sub, mul, div, fdiv, power
 #we can return multiple values together.
tpl=calculator(50,6)
for x in tpl:
    print(x)
```

```
56
44
300
8.333333333333334
8
15625000000
>>>
```

Q). write a program that shows the passing of number of keyword arguments together using *Kwargs parameter?

Sol).

```
def myFun(**kwargs):
    for key, value in kwargs.items():
        print ("%s = %s" %(key, value))
# Driver code
 myFun(st_id =10, name ='Avanish', course='MCA')
```

Output-:

```
st_id = 10
name = Avanish
course = MCA
>>>
```

Q). define Scope and Lifetime of Variables in Python? (v. important)

Sol).

A variable isn't visible everywhere and alive every time. We study this in functions because the scope and lifetime for a variable depend on whether it is inside a function.

A variable's scope tells us where in the program it is visible. A variable may have local or global scope.

a) Local Scope- A variable that is declared inside a function has a local scope. In other words, it is local to that function.
Local variables will only have their existence inside a function. Which means you cannot use a local variable outside a function.

Ex).

```
def func3():
    x=7       # local variable
    print(x)
func3()
```

b) Global Scope-:
When you declare a variable outside python function, or inside a function with global keyword then that variable will be called as global variable. It means that it is visible everywhere within the program. Global variable can be accessed inside a function or outside a function.

Ex).

```
y=7 # global variable
def func4():
    print(y)
func4()
```

---

note-: However, you can't change its value from inside a local scope (here, inside a function). To do so, you must declare it global inside the function, using the 'global' keyword.

Ex).

```
def func4():
    global y# global variable inside a function
    y+=1
    print(y)
func4()
```

Q). example of local variable, global variable in python?

Sol).

```
i = 10       # global variable
m = 20       # global variable
def my_name(name):
    j=5      ## local variable
    k=7      ## local variable
    print(j,k)       #priting local variable inside function
    print(i)         #priting global variable inside function
    print("hello my name is: ",name)
my_name("Avanish Singh")
print(i,m)       # priting global variable outside the function
print(j,k)       #priting local variable outside function
#we cannot use local variables outside the function.
```

Output-:

```
5 7
10
hello my name is:  Avanish Singh
10 20
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\first.py", line 13, in <module>
    print(j,k)       #priting local variable outside function
NameError: name 'j' is not defined
>>>
```

Q). write a program that shows how to change the global variable value inside function? (important)

Sol).

```
i = 10
def my_name(name):
    global i
    i = i+23
    print(i)
    print("hello my name is: ",name)
my_name("Avanish Singh")
print(i)
```

Output-:

```
33
hello my name is:  Avanish Singh
33
>>>
```

Q). define recursion in python? (important)

Sol).

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily.

Q). write a function program of python to get the factorial result using recursion. (important)

Sol).

```
# n! = n*n-1!
def get_fact(n):
    if n==0:
        return 1
    elif n==1:
        return 1
    else:
        return n* get_fact(n-1)
num = int(input("Enter any number to get factorial result: "))
print(get_fact(num))
```

Output-:

```
Enter any number to get factorial result: 6
720
>>> |
```

Q). write a python program to print Fibonacci series using recursion in function. (important)

Sol).

```
# 0,1,1,2,3,5,8.....
def fibonacci(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
turm = int(input("Enter the turm upto the series you want: "))
for i in range(turm):
    print(fibonacci(i))
```

Output-:

```
Enter the turm upto the series you want: 10
0
1
1
2
3
5
8
13
21
34
>>> |
```

Q). define Python Lambda or anonymous function? (important)

Sol).

- In Python, an anonymous function is a function that is defined without a name.
- While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.
- Hence, anonymous functions are also called lambda functions.
- A lambda expression in Python allows us to create anonymous python function, and we use the 'lambda' keyword for it. The following is the syntax for a lambda expression.
- A lambda function can take any number of arguments, but can only have one expression.

- it is another way of creating the function.
- lambda is single line function so it is called anonymous function.
- The power of lambda is better shown when you use them as an anonymous function inside another function.

Syntax-:

lambda arguments: expression

Ex).

myvar = lambda a, b: (a*b)

myvar(3,5)

Q). write a program that shows the difference between simple function program and lambda function program?

Sol).

```
# simple function program
def multiply(a,b):
    c = a*b
    print(c)
multiply(5,8)
```

```
#lambda function program
multiply = lambda a,b:a*b
print(multiply(5,8))
```

Output-:

```
40
>>>
```

Q). write a function program that shows the use of fstring in python?

Sol).

```
x = 20
y = 30
som = x + y
prod = x * y
print(f'{x} + {y} = {som}; {x} * {y} = {prod}')
```

Output-:

```
20 + 30 = 50; 20 * 30 = 600
>>>
```

Q). write a function program that sort only the first elements of list of lists.

Sol).

```
a = [[5,6,3],[3,6],[1,4,6]]
a.sort(key=lambda a:a[0])
print(a)
```

Output-:

```
[[1, 4, 6], [3, 6], [5, 6, 3]]
>>>
```

Q). what is map function in python?

Sol).

Python's **map()** is a built-in function that allows you to process and transform all the items in an iterable without using an explicit for loop, a technique commonly known as mapping. map() is useful when you need to apply a **transformation function** to each item in an iterable and transform them into a new iterable. map() is one of the tools that support a functional programming style in Python.

Syntax-:

```
map(function, iterable(s))
```

Ex-:

```
my_strings = ['a', 'b', 'c', 'd', 'e']
my_numbers = [1,2,3,4,5]
results = list(map(lambda x, y: (x, y), my_strings, my_numbers))
print(results)
```

Example-:

```
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
>>>
```

Q). write a program that check how may elements of a list are staring with 'A' character?

Sol).

```
fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]
map_object = map(lambda s: s[0] == "A", fruit)
print(list(map_object))
```

Output-:

```
[True, False, False, True, False]
>>>
```

Note-: we can also print those fruit names by using for loop with map_object

Ex-: for i in map_object:

print(i)

Q). What is filter function in python?

Sol). The filter() function returns an iterator were the items are filtered through a function to test if the item is accepted or not.

The filter() function is similar to a for-loop in Python but is much fast as a built-in function.

It takes as an argument a *function* and an *iterable* and applies the past function to each element of the iterable. Once this is done, it returns an *iterable*.

Syntax-:

filter(function, iterable(s))

Q). write a program that shows the use of filter method with lambda expression by printing only those elements from a list which are divisible by 2?

Sol).

```
my_list = [1, 5, 4, 6, 8, 11, 3, 12, 15, 17, 18, 66, 55]
new_list = list(filter(lambda x: (x%2 == 0) , my_list))
print(new_list)
```

Output-:

```
[4, 6, 8, 12, 18, 66]
>>>
```

Q). write a program to check the palindrome strings using filter function

Sol).

```
dromes = ("demigod", "rewire", "madam", "freer", "anutforajaroftuna", "kiosk")
palindromes = list(filter(lambda word: word == word[::-1], dromes))
print(palindromes)
```

Output-:

```
['madam', 'anutforajaroftuna']
>>>
```

Q). write a function program that calculate upper and lower character from any statement?

Sol).

```
def string_test(s):
    d={"UPPER_CASE":0, "LOWER_CASE":0}
    for c in s:
        if c.isupper():
            d["UPPER_CASE"]+=1
        elif c.islower():
            d["LOWER_CASE"]+=1
        else:
            pass
    print ("Original String : ", s)
    print ("No. of Upper case characters : ", d["UPPER_CASE"])
    print ("No. of Lower case Characters : ", d["LOWER_CASE"])
string_test('Python has the richest Library')
```

Output-:

```
Original String :  Python has the richest Library
No. of Upper case characters :  2
No. of Lower case Characters :  24
>>>
```

Q). write a function program that identify unique numbers inside a list?

Sol).

```
def unique_list(li):
    x = []
    for a in li:
        if a not in x:
            x.append(a)
    return x
print(unique_list([1,2,3,3,3,3,4,5]))
```

Output-:

```
[1, 2, 3, 4, 5]
>>>
```

Q). Write a Python program to detect the number of local variables declared in a function?

Sol).

```
a = 10
def abc():
    global m
    m = 'Avanish'
    x = 1
    y = 2
    str1= "Hello Python Lovers"
    print("Python Series")
print(abc.__code__.co_nlocals)
```

output-:

```
3
>>>
```

Q). Write a Python program to find intersection of two given arrays using Lambda.

Sol).

```
array_nums1 = [1, 2, 3, 5, 7, 8, 9, 10]
array_nums2 = [1, 2, 4, 8, 9]
print("Original arrays:")
print(array_nums1)
print(array_nums2)
result = list(filter(lambda x: x in array_nums1, array_nums2))
print ("\nIntersection of the said arrays: ",result)
```

Output-:

```
Original arrays:
[1, 2, 3, 5, 7, 8, 9, 10]
[1, 2, 4, 8, 9]

Intersection of the said arrays:  [1, 2, 8, 9]
>>>
```

Files processing

Q). define files? (important)

Sol).

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).

When we want to read or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Q) define file operations in python? (important)
Sol).

- Open a file
- Read or write (perform operation)
- Close the file

Other operations include:

- Rename
- Delete

Q). define open method of file with syntax? (important)
Sol).
Python has an in-built function called open() to open a file. It is also used to create a file as well as doing so many operations in file.

It takes a minimum of one argument as mentioned in the below syntax. The open method returns a file object which is used to access the write, read and other in-built methods.

Syntax:

```
file_object = open(file_name, mode)
```

Here, file_name is the name of the file or the location of the file that you want to open, and file_name should have the file extension included as well. Which

means in **test.txt** – the term test is the name of the file and .txt is the extension of the file.
The mode in the open function syntax will tell Python as what operation you want to do on a file.

We use **open ()** function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use **open()** function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: **open(filename, mode)**.

Q). example of open function to create a file? (important)
Sol

```
# a file named "ys", will be opened with the reading mode.
file = open('c:/python/ys.txt', 'r')
# This will print every line one by one in the file
for each in file:
    print (each)
```

The open command will open the file in the read mode and the for loop will print each line present in the file.

Q). discuss with command for open function in python?

Sol).

with is a command which is used with open to perform all the operations of file. Instead of using previous syntax you can also use with command syntax to perform all the operations of file as given below-:

syntax-:

with open('file_name', 'mode_of_file') as variable_name:

Ex).

```
with open("D:/Documents/file.text", "r") as f:
    print(f.read())
```

Q). define python file modes? (important)

Sol).

- 'r' – Read Mode: Read mode is used only to read data from the file.

- ‘w' – Write Mode: This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.
- ‘a' – Append Mode: Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
- ‘r+' – Read or Write Mode: This mode is used when we want to write or read the data from the same file.
- ‘a+' – Append or Read Mode: This mode is used when we want to read data from the file or append the data into the same file.

Note: The above-mentioned modes are for opening, reading or writing text files only.

While using binary files, we have to use the same modes with the letter **‘b'** at the end. So that Python can understand that we are interacting with binary files.

- ‘wb’ – Open a file for write only mode in the binary format.
- ‘rb’ : Open a file for the read-only mode in the binary format.
- ‘ab’: Open a file for appending only mode in the binary format.
- ‘rb+’: Open a file for read and write only mode in the binary format.
- ‘ab+’: Open a file for appending and read-only mode in the binary format.

Q). is it compulsory to provide the mode with open function in python?

Sol).

No, it is not compulsory to provide the mode of file. One must keep in mind that the mode argument is not mandatory. If not passed, then Python will assume it to be “ **r** ” by default. Let’s look at this program and try to analyse how the read mode works:

Q). discuss all Python File Methods? (important)

Sol).

Function	Explanation
open()	To open a file
close()	Close an open file
fileno()	Returns an integer number of the file
read(n)	Reads ‘n’ characters from the file till end of the file

Function	Explanation
readable()	Returns true if the file is readable
readline()	Read and return one line from the file
readlines()	Reads and returns all the lines from the file
seek(offset)	Change the cursor position by bytes as specified by the offset
seekable()	Returns true if the file supports random access
tell()	Returns the current file location
writable()	Returns true if the file is writable
write()	Writes a string of data to the file
writelines()	Writes a list of data to the file

Q). write a program to open a file in read and write mode? **(important)**
Sol).

with open("C:/Documents/Python/test.txt", "r+") as file:

In the above example, we are opening the file named 'test.txt' present at the location 'C:/Documents/Python/' and we are opening the same file in a read-write mode which gives us more flexibility.

Q)) write a function program to read the entire file after creating function.
Sol).

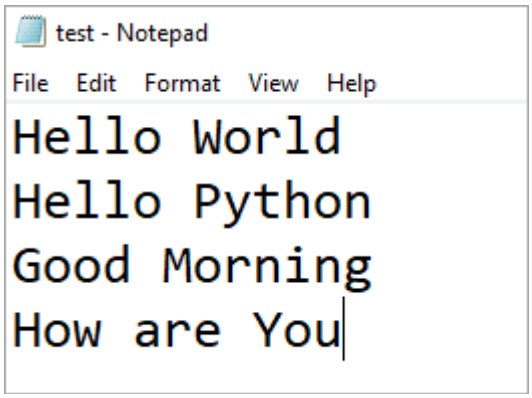
```
def file_read(fname):
    txt = open(fname)
    print(txt.read())
file_read("C:/Users/AVIRA/Desktop/avisingh.txt")
```

Q). explain the methods to read any file.? **(important)**
Sol). **There are three ways in which we can read the files in python-:**

- read([n])
- readline([n])
- readlines()

Here, n is the number of bytes to be read.

First, let's create a sample text file as shown below.



Now let's observe what each read method does:

```
my_file = open('C:/Documents/Python/test.txt', 'r')
print(my_file.read(5))
```

Output:

Hello

Here we are opening the file test.txt in a read-only mode and are reading only the first 5 characters of the file using the my_file.read(5) method.

Example of read():

```
with open('C:/Documents/Python/test.txt', 'r') as my_file:
    print(my_file.read())
```

Output:

Hello World
Hello Python
Good Morning

Here we have not provided any argument inside the read() function. Hence it will read all the content present inside the file.

Example of readline():

```
my_file = open('C:/Documents/Python/test.txt', 'r')
print(my_file.readline())
```

Output:
Hello World

readline() will read only first line if we do not provide any argument to the method. Suppose if you provide any method then it will print only that line.

If you are giving the argument to the readline() then it will return only that length of characters containing the spaces, which you have provided as an argument.

Example of readlines:

```
my_file = open('C:/Documents/Python/test.txt', 'r')
print(my_file.readlines())
```

Output:
['Hello World\n', 'Hello Python\n', 'Good Morning']

Here we are reading all the lines present inside the text file including the newline characters.

Q). write a program in python to Read a specific line from a File. **(important)**
Sol).

```
line_number = 3
fo = open('C:/Users/avira/Documents/python/text.txt', 'r')
currentline = 1
for line in fo:
    if(currentline == line_number):
        print(line)
        break
    currentline = currentline +1
```

Output:

```
hope you are doing well

>>> |
```

In the above example, we are trying to read only the 4th line from the 'test.txt' file using a **"for loop"**.

Q). discuss write methods in python file handling?

Sol).

In order to write data into a file, we must open the file in write mode. We need

to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.

We have two methods for writing data into a file as shown below.

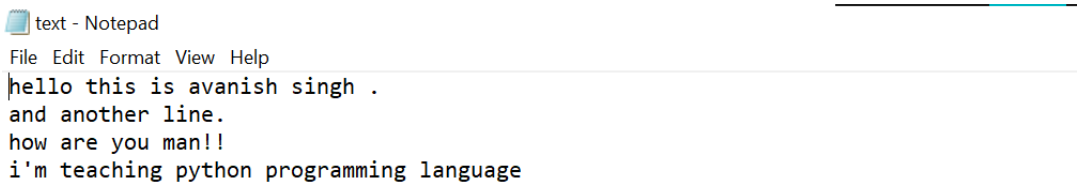
- write(string)
- writelines(list)

Q). python program to open a file in write mode and write some content to that file? **(important)**

Sol).

```
with open('C:/Users/avira/Documents/python/text.txt', 'w') as f:
    f.write ('hello this is avanish singh .\n')
    f.write ('and another line.\n')
    f.write('how are you man!!\n')
    f.write("i'm teaching python programming language")
```

Output-:



Q) write a program to writing into a file that does not exist? **(important)**
Sol)

```
with open('C:/Users/avira/Documents/python/text2.txt', 'xt') as f:
    f.write ('Hello, This is my new file.\n')
```

output-:

Q). write a program to write text data of list type into a file name as test.txt?
Sol).

```
fruits = ['Apple\n', 'Orange\n', 'Grapes\n', 'Watermelon']
my_file = open('C:/Users/avira/Documents/python/text2.txt', 'w')
my_file.writelines(fruits)
```

Output-:

The above code writes a **list of data** into the 'test.txt' file simultaneously.

Q) discuss how to Write and Read Data to a Binary File with an example?
Sol).
Binary files store data in the binary format (0's and 1's) which is understandable by the machine. So when we open the binary file in our machine, it decodes the data and displays in a human-readable format.

Example:
#Let's create some binary file.

```
my_file = open('C:/Documents/Python/bfile.bin', 'wb+')
message = 'Hello Python'
file_encode = message.encode('ASCII')
my_file.write(file_encode)
my_file.seek(0)
bdata = my_file.read()
print('Binary Data:', bdata)
ntext = bdata.decode('ASCII')
print('Normal data:', ntext)
```

In the above example, first we are creating a binary file **'bfile.bin'** with the read and write access and whatever data you want to enter into the file must be encoded before you call the write method.
Also, we are printing the data without decoding it, so that we can observe how the data exactly looks inside the file when it's encoded and we are also printing the same data by decoding it so that it can be readable by humans.

Q) Write a Python program to read a file line by line and store it into a list?
Sol).

```
def file_read(fname):
    with open(fname) as f:
        content_list = f.readlines()
        print(content_list)
file_read("C:/Users/avira/Documents/python/text.txt")
```

Output-:

```
['hello this is avanish singh .\n', 'and another line.\n', 'how are you man!!\
n', "i'm teaching python programming language"]
>>>
```

Q)) Write a Python program to count the number of lines in a text file.
Sol).

```
def file_lengthy(fname):
    with open(fname) as f:
        for i, l in enumerate(f):
            pass
        return i + 1
x = file_lengthy("C:/Users/avira/Documents/python/text.txt")
print("Number of lines in the file:",x)
```

output-:

```
Number of lines in the file: 4
>>>
```

Q) Write a Python program to count the frequency of words in a file.
Sol). **(important)**

```
from collections import Counter
def word_count(fname):
    with open(fname) as f:
        return Counter(f.read().split())
x = word_count("C:/Users/avira/Documents/python/text.txt")
print("Number of words in the file :",x)
```

Output-:

```
Number of words in the file : Counter({'hello': 1, 'this': 1, 'is': 1, 'avanis
h': 1, 'singh': 1, '.': 1, 'and': 1, 'another': 1, 'line.': 1, 'how': 1, 'are'
: 1, 'you': 1, 'man!!': 1, "i'm": 1, 'teaching': 1, 'python': 1, 'programming'
: 1, 'language': 1})
>>>
```

Q). discuss append mode in python? **(important)**

Sol).

Append mode(a) in python is used to append the data into a file. It always appends (add) the data at the end of the file. It also does not remove the previous data. It just adds new data at the end of file.

To append data into a file we must open the file in 'a+' mode so that we will have access to both the append as well as write modes.

Q). write a program to append a data into the existing file.

Sol).

```
my_file = open("C:/Users/avira/Documents/python/text.txt",'a+')
my_file.write ('\nStrawberry')
my_file.close()
```

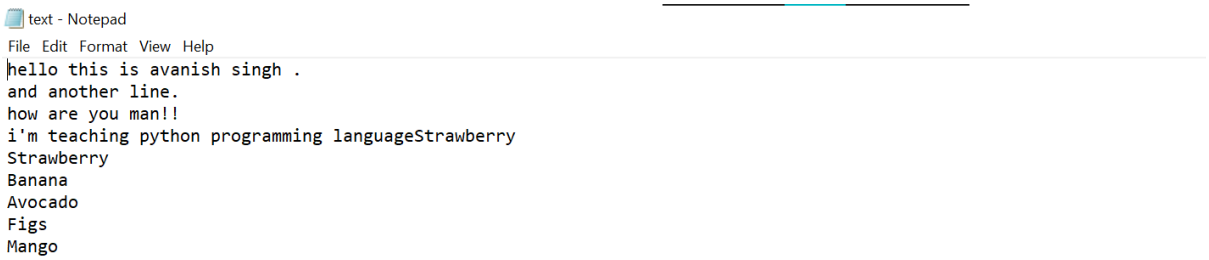
Output-:

```
hello this is avanish singh .
and another line.
how are you man!!
i'm teaching python programming languageStrawberry
Strawberry
```

Q). write a program to append the list data into a file.

Sol).

```
fruits = ['\nBanana', '\nAvocado', '\nFigs', '\nMango']
my_file = open("C:/Users/avira/Documents/python/text.txt",'a+')
my_file.writelines(fruits)
my_file.close()
```



Q). discuss File I/O Attributes in python? **(important)**
Sol).

Attribute	Description
Name	Return the name of the file
Mode	Return mode of the file
Encoding	Return the encoding format of the file
Closed	Return true if the file closed else returns false

Q). write a python program to print the name, mode, encoding format of the file. Also check weather file is closed or not? and check weather file is readable and writable or not?
Sol).

```
my_file = open("C:/Users/avira/Documents/python/text.txt",'a+')
print('What is the file name? ', my_file.name)
print('What is the file mode? ', my_file.mode)
print('What is the encoding format? ', my_file.encoding)
print('Is file readable: ?', my_file.readable())
print('Is file writeable: ?', my_file.writable())
print('File no:', my_file.fileno())


print('Is File closed? ', my_file.closed)
my_file.close()
print('Is File closed? ', my_file.closed)
```

Output-:

```
What is the file name?  C:/Users/avira/Documents/python/text.txt
What is the file mode?  a+
What is the encoding format?  cp1252
Is file readable: ? True
Is file writeable: ? True
File no: 3
Is File closed?  False
Is File closed?  True
>>>
```

Q). discuss on closing a Python File? **(important)**
Sol).
In order to close a file, we must first open the file. In python, we have an in-built method called close() to close the file which is opened.

Whenever you open a file, it is important to close it, especially, with write method. Because if we don't call the close function after the write method then whatever data we have written to a file will not be saved into the file.

```
my_file = open('C:/Documents/Python/test.txt', 'w')
my_file.write('Hello World')
my_file.close()
```

Q). discuss Python Rename or Delete File?
Sol).
Python provides us with an "os" module which has some in-built methods that would help us in performing the file operations such as renaming and deleting the file.

In order to use this module, first of all, we need to import the "os" module in our program and then call the related methods.

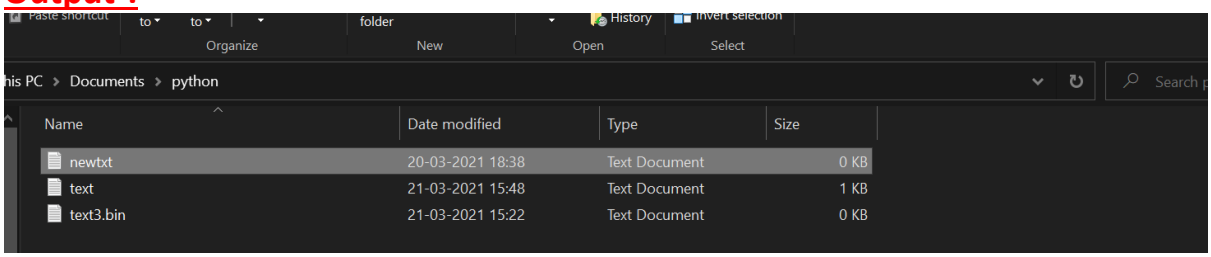
rename() method:
This rename() method accepts two arguments i.e. the current file name and the new file name.

Syntax:
os.rename(current_file_name, new_file_name)

Q). write a program to rename a file? **(important)**
Sol)

```
import os
os.rename('C:/Users/avira/Documents/python/text2.txt',
          'C:/Users/avira/Documents/python/newtxt.txt')
```

Output-:



remove() method:

We use the remove() method to delete the file by supplying the file name or the file location that you want to delete.

Syntax:

os.remove(file_name)

Scope and Modules

Q). Define namespaces in python? **(important)**
Sol).
A namespace is a container where names are mapped to objects, they are used to avoid confusions in cases where same names exist in different namespaces. They are created by modules, functions, classes etc.

The namespace word itself can be broken down as **name (which is a unique identifier) + space (which means relating to scope)**. The name can be anything from method name to variable name and space depends upon the location from where we are trying to access a variable or a method.

Q). Write down the types of namespaces? **(important)**

Sol).

1. Built-in Namespace

This namespace includes functions and exception names that are built-in in the Python.

2. Global Namespace

This namespace includes names from the modules that are imported in the project. It is created when we include the module and it lasts until the script ends.

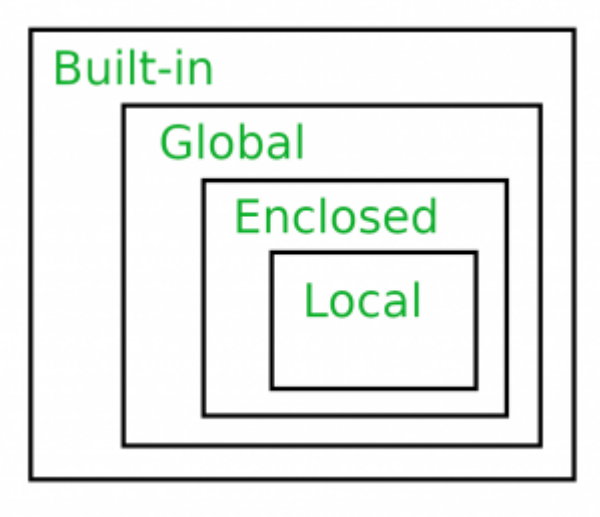
3. Local Namespace

The local names inside the function comes under a local namespace. This namespace is created when the function is called and the scope ends when the value is returned.

Q). describe the LEGB rule with example? **(important)**
Sol).
In Python, the **LEGB rule** is used to decide the order in which the namespaces are to be searched for scope resolution.
The scopes are listed below in terms of hierarchy (highest to lowest/narrowest to broadest):

- Local(L): Defined inside function/class
- Enclosed(E): Defined inside enclosing functions (Nested function concept)

- Global(G): Defined at the uppermost level
- Built-in(B): Reserved names in Python built-in modules



Q). example that shows the use of Local, Enclosed and Global Scopes?
Sol).

```
pi = "this is global variable"
def outer():
    pi = "This is local variable"
    print(pi)
    def inner():
        nonlocal pi
        pi = "this is enclosed variable" #Enclosed Namespace
        print(pi)
    inner()
outer()
print(pi)
```

Output:

```
This is local variable
this is enclosed variable
this is global variable
>>>
```

Note-: When outer *()* is executed, *inner ()* and consequently the *print* functions are executed, which print the value the enclosed *pi* variable. The statement in line 10 looks for variable in local scope of *inner*, but does not find it there. Since *pi* is referred with the *nonlocal* keyword, it means that *pi* needs to be accessed from the *outer* function (i.e. the outer scope). To summarize, the *pi* variable is not found in local scope, so the higher scopes are looked up. It is found in both enclosed and global scopes. But as per the LEGB hierarchy, the enclosed scope variable is considered even though we have one defined in the global scope.

Q). example that shows the use of Local, Enclosed, Global and Built-in Scopes in python.
Sol).

```
from math import pi
pi = 'global pi variable'
def outer():
    pi = 'local pi variable'
    print(pi)
    def inner():
        nonlocal pi
        pi = 'Enclosed pi variable'
        print(pi)
    inner()

outer()
print(pi)
```

Output:

```
local pi variable
Enclosed pi variable
global pi variable
>>>
```

Q). define the python module with examples?

Sol).

A module is a file containing Python definitions and statements, So the python files are also called module. A module can define functions, classes and variables. They are its attributes. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

Module files contain so many attributes, by importing these methods a programmer can do so many things with ease in less line of coding. A python file called hello.py has the module name of hello that can be imported into other python files.

Some modules are built in so we need not to install then into or python Scripts otherwise there are many modules that we need to install into our python Script before using them into our python file.

For installing these modules into python scripts, we use the below command -:

pip install module_name

Ex-: pip install pillow

Ex-: pip install googletrans

Q). write down the syntax which we use to import the module.
Sol).
To access any module into our python file there are two things we must do-:

1. We should install that module into our python script if it is not built in module (i.e. if it is not preinstalled in python script).
2. Once it is installed in our python script we must import it into our python file.

For importing a module in our python file we can use any of the three ways -:

1. import Module_name1, Module_Name2, Module_name3, …, etc.
   above way we can import many modules into our python file in one line of code.
   Ex).
   ```
   import tkinter, sys, random,
   ```

2. from module_name import attribute1, attribute2, attribute3, …., etc
   above way we can import so many attributes from any modules.
   Ex).
   ```
   from math import sqrt, factorial, sum
   ```

3. from module import *
   above syntax will allow you to use all the attribute from the imported module. Here * symbol is defining the all attributes.
   Ex).
   ```
   from tkinter import *
   ```
   It will import all the attributes of tkinter module.

Q). write a program that finds square root and factorial of any number using built-in module of python.
Sol).

```
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

Output:

```
4.0
720
>>>
```

Q). write a program that print all the attributes (modules, variables and functions) of any module.

Sol).

The dir() built-in function returns a sorted list of strings containing the names defined by a module. The list contains the names of all the modules, variables and functions that are defined in a module.

```
#  Import built-in module  random
import random
print(dir(random))
```

Output-:

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'Syst
emRandom', 'TWOPI', '_Sequence', '_Set', '__all__', '__builtins__', '__cached_
_', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__'
, '_accumulate', '_acos', '_bisect', '_ceil', '_cos', '_e', '_exp', '_inst', '
_log', '_os', '_pi', '_random', '_repeat', '_sha512', '_sin', '_sqrt', '_test'
, '_test_generator', '_urandom', '_warn', 'betavariate', 'choice', 'choices',
'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvar
iate', 'normalvariate', 'paretovariate', 'randint', 'random', 'randrange', 'sa
mple', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariat
e', 'weibullvariate']
>>>
```

Q) Write a program that illustrates attributes of math module. **(important)**

```
# importing built-in module math
import math
# using square root(sqrt) function contained
print(math.sqrt(25))
# using pi function contained in math module
print(math.pi)
# Sine of 2 radians
print(math.sin(2))

# Cosine of 0.5 radians
print(math.cos(0.5))

# Tangent of 0.23 radians
print(math.tan(0.23))
print(math.factorial(4))
```

Output-:

```
5.0
3.141592653589793
0.9092974268256817
0.8775825618903728
0.23414336235146527
24
>>>
```

Q). write a program that illustrate the use of random module in python. (important)

Sol).

```
import random
# printing random integer between 0 and 5
print(random.randint(0, 5))
# print random floating point number between 0 and 1
print(random.random())
# random number between 0 and 100
print(random.random() * 100)

List = [1, 4, True, 800, "python", 27, "hello"]
# using choice function in random module for choosing
# a random element from a set such as a list
print(random.choice(List))
```

Output-:

```
0
0.6677476952788014
33.00796678393228
python
>>> 
```

Object Oriented programming

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviours are bundled into individual **objects**.

For instance, an object could represent a person with **properties** like a name, age, and address and **behaviours** such as walking, talking, breathing, and running. Or it could represent an email with properties like a recipient list, subject, and body and behaviours like adding attachments and sending.

Python is an object-oriented programming language.

Almost everything in Python is an object, with its properties and methods.

Class-:

A class is a blueprint for how something should be defined.

A straight forward answer to this question is- A class is a collection of objects. Unlike the primitive data structures, classes are data structures that the user defines. They make the code more manageable.

Let's see how to define a class below-

```
class class_name:

    class body
```

We define a class with a keyword "class" following the class_name and semicolon. And we consider everything you write under this after using indentation as its body.

Ex-:

```
class Myclass:
    pass
```

Object-:

While the class is the blueprint, an **instance** is an object that is built from a class and contains real data. The object is an entity that has state and behaviour.

When we define a class only the description or a blueprint of the object is created. There is no memory allocation until we create its **object**.
The **objector instance** contains real data or information.

Instantiation is nothing but creating a new object/instance of a class

how to create an object-:

instace_variable/object_variable = class_name()

Q). take a example to create a class, create an object and print the instance variable.

```
class MyClass: # class creation
    x = 5        # instance variable

p1 = MyClass()   # creating an object/creating an instace of class
print(p1.x)      # accessing the instance variable inside a class
```

Output-:

```
5
>>>
```

Type of variables inside class and object-:

Instance variable-: a variable declared inside a class or outside the class. They are also called the variable of object. Since they are object variable so they will be accessed using object name.

We can access instance anywhere inside or outside the class.

If you have created constructor then you will access the instance variable with self parameter inside the class and using instance name outside the class.

Local variable-:

Local variables are those variables which are declared inside a function (first time). These variables can only be used inside a function. You can not use/access them outside that function.

Class variable-:

Class variable are those variables that are declared inside a class with the class name. these variables can be access inside or outside a class using class name.

Note-:

Instance variables are variables whose value is assigned inside a constructor or method with self, whereas class variables are variables whose value is assigned in the class.

Constructor in python-:

Constructor is a special function that has the same name as the class.

Python has a special name of the constructor which is __init__.

__init__ is a method which is immediately and automatically called after an instance has been created. This name is fixed and it is not possible to choose another name. __init__ is one of the so-called magic methods, we will get to know it with some more details later. The __init__ method is used to initialize an instance. There is no explicit constructor or destructor method in Python

Syntax of constructor declaration:

```
def __init__(self):
    # body of the constructor
```

Note-: we create constructor to initialize our instance variables inside a class.

Constructor can take any number of parameters. The first parameter is the copy of the name of instance/object inside the class (it means constructors first variable is nothing but just a name that is another reference of your object).

Self Parameter-:

It is the first parameter of any constructor. You can give it any name but this will be the variable using which you can access of the instance variables inside a class. Because self is just a copy of your instance variable.

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Types of constructors :

- **default constructor:** The default constructor is simple constructor which doesn't accept any arguments. If you did not create any constructor in your program it will automatically create by the interpreter.
  Syntax-:
  def __init__():
- **parameterized constructor:** constructor with parameters is known as parameterized constructor. The parameterized constructor takes its first argument as a reference to the instance being constructed known as self and the rest of the arguments are provided by the programmer.

Note -: you cannot overload the constructor in python. If you try, it will automatically follow python override property.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Output-:

```
John
36
>>>
```

EX-:

```
class Employee:
    def __init__(self, name, id):
        self.id = id
        self.name = name

    def display(self):
        print("ID is:", self.id,"and name is:", self.name)


emp1 = Employee("John", 101)
emp2 = Employee("David", 102)

# accessing display() method to print employee 1 information

emp1.display()

# accessing display() method to print employee 2 information
emp2.display()
```

Output-:

```
ID is: 101 and name is: John
ID is: 102 and name is: David
>>>
```

Ex-:

```
class Car:
    car_type = "Sedan"

    def __init__(self, name, mileage):
        self.name = name
        self.mileage = mileage

    def description(self):
        return f"The {self.name} car gives the mileage of {self.mileage}km/l"

    def max_speed(self, speed):
        return f"The {self.name} runs at the maximum speed of {speed}km/hr"

obj2 = Car("Honda City",24.1)
print(obj2.description())
print(obj2.max_speed(150))
```

Output-:

```
The Honda City car gives the mileage of 24.1km/l
The Honda City runs at the maximum speed of 150km/hr
>>>
```

Ex-: you can name anything to the first parameter of the constructor

```
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age

    def myfunc(abc):
        print("Hello my name is:" ,abc.name)
        print("my age is:",abc.age)

p1 = Person("John", 36)
p1.myfunc()
```

Output-:

```
Hello my name is: John
my age is: 36
>>>
```

Modify an instance variable-:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.age = 40

print(p1.age)
```

Output-:

```
40
>>>
```

Object oriented properties-:

1. Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class.

Parent class is the class being inherited from, also called base class.

Child class is the class that inherits from another class, also called derived class.

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)

x = Student("Deepak", "Chauhan")
x.printname()
```

Output-:

```
Deepak Chauhan
>>>
```

super() Function

Python also has a super() function that will make the child class inherit all the methods and properties from its parent. By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent.

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)

class Student(Person):
    def __init__(self, fname, lname, year):
        super().__init__(fname, lname)
        self.graduationyear = year

    def welcome(self):
        print("Welcome", self.firstname, self.lastname,
              "to the class of", self.graduationyear)

x = Student("Mike", "Olsen", 2019)
x.welcome()
```

Output-:

```
Welcome Mike Olsen to the class of 2019
>>>
```

2. Data abstraction

We use Abstraction for hiding the internal details or implementations of a function and showing its functionalities only. This is similar to the way you know how to drive a car without knowing the background mechanism. Or you know how to turn on or off a light using a switch but you don't know what is happening behind the socket.

3. Polymorphism

This is a Greek word. If we break the term Polymorphism, we get “poly”-many and “morph”-forms. So, Polymorphism means having many forms. In OOP it refers to the functions having the same names but carrying different functionalities.

```
class Audi:
    def description(self):
        print("This the description function of class AUDI.")

class BMW:
    def description(self):
        print("This the description function of class BMW.")
audi = Audi()
bmw = BMW()
for car in (audi,bmw):
    car.description()
```

Output-:

```
This the description function of class AUDI.
This the description function of class BMW.
>>>
```

4. Encapsulation

Encapsulation, as I mentioned in the initial part of the article, is a way to ensure security. Basically, it hides the data from the access of outsiders. Such as if an organization wants to protect an object/information from unwanted access by clients or any unauthorized person then encapsulation is the way to ensure this.

You can declare the methods or the attributes protected by using a single underscore (_) before their names. Such as- self._name or def _method( ); Both of these lines tell that the attribute and method are protected and should not be used outside the access of the class and sub-classes but can be accessed by class methods and objects.

Though Python uses ‘ _ ‘ just as a coding convention, it tells that you should use these attributes/methods within the scope of the class. But you can still access the variables and methods which are defined as protected, as usual.

Now for actually preventing the access of attributes/methods from outside the scope of a class, you can use "**private members**". In order to declare the attributes/method as private members, use double underscore (__) in the prefix. Such as – self.__name or def __method(); Both of these lines tell that the attribute and method are private and access is not possible from outside the class.

```
class car:
    def __init__(self, name, mileage,amt):
        self._name = name                #protected variable
        self.mileage = mileage
        self.__amt = amt    # private variable

    def description(self):
        return f"The {self._name} car gives the mileage of {self.mileage}km/l"
obj = car("BMW 7-series",39.53,1000000)

#accessing protected variable via class method
print(obj.description())

#accessing protected variable directly from outside
print(obj._name)
print(obj.mileage)
#accessing private variable directly from outside
print(obj.__amt)
# but you can access private variable outside a class by doing mangling
print(obj._car__amt)      #mangled name
```

Output-:

```
The BMW 7-series car gives the mileage of 39.53km/l
BMW 7-series
39.53
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\first.py", line 19, in <module>
    print(obj.__amt)
AttributeError: 'car' object has no attribute '__amt'
>>>
```

Note-:

When we tried accessing the private variable using the description() method, we encountered no error. But when we tried accessing the private variable directly outside the class, then Python gave us an error stating: car object has no attribute '__name'.

You can still access this attribute directly using its mangled name. **Name mangling** is a mechanism we use for accessing the class members from outside. The Python interpreter rewrites any identifier with "__var" as

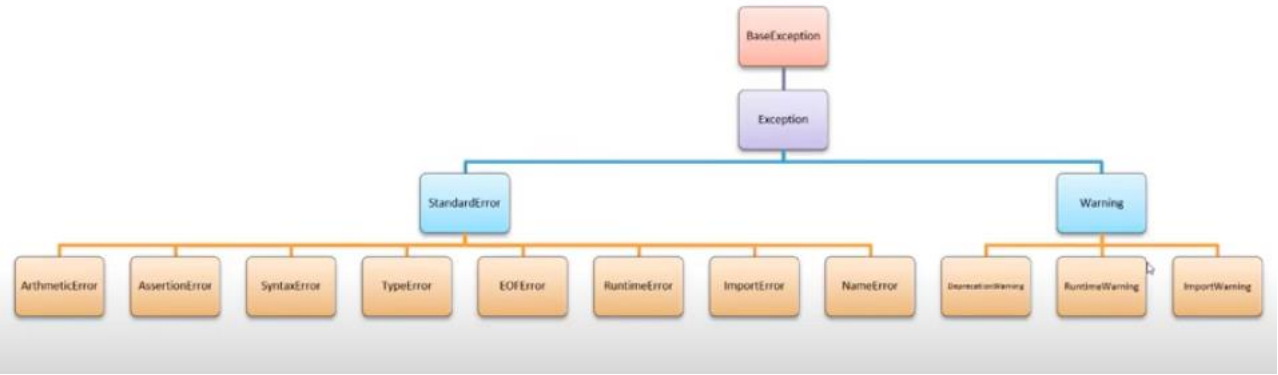
"_ClassName__var". And using this you can access the class member from outside as well.

Exception Handling

Exception-: Exception is an event which will occur during the execution of program that disrupts the normal flow of the program's instruction. An exception is run time error that can be handle by the programmer. All exceptions are represented as the classes in python.

Type of exception-:

1. Built in exception -: Exception which are already available in python language. Base class for all built in exceptions is BaseException class.
2. User defined exception-: a program can define his own exception called as user defined exception.



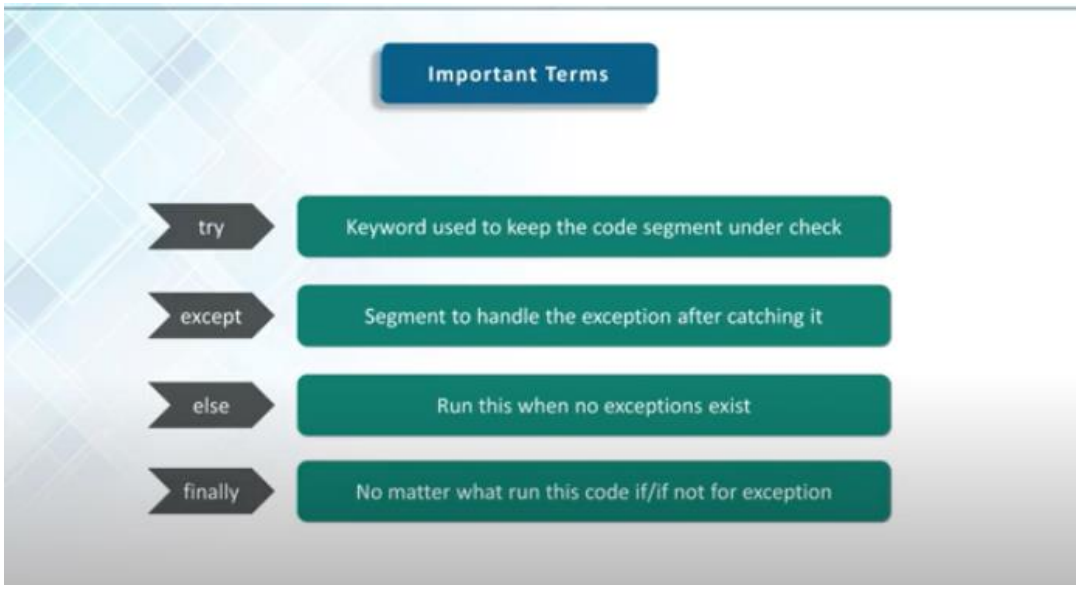
Need of exception handling-:

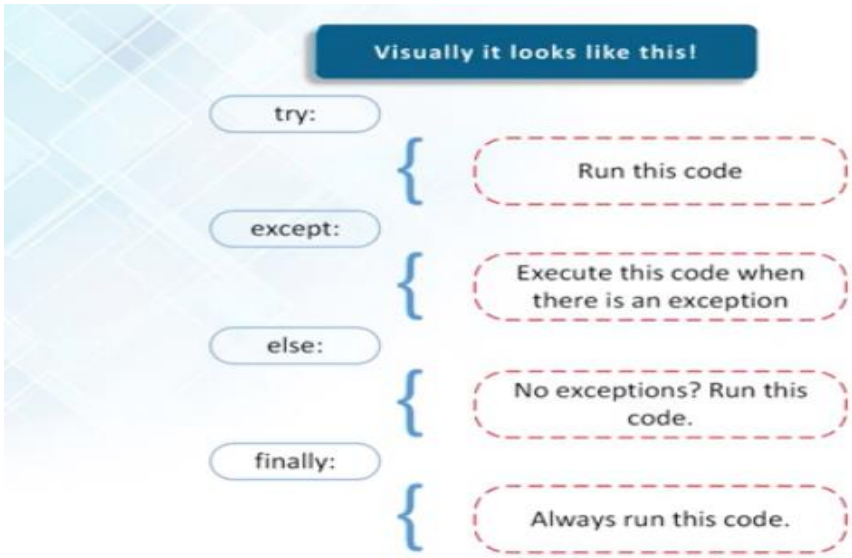
5. When an exception occurs, the program terminates suddenly.
6. Suddenly termination of program may corrupt the program.
7. Exception may cause data loss from the database or file.

Exception Handling-: process of responding to the occurrence during computation of exceptional condition requiring special processing-often changing the normal flow of the program execution.

→

(try block) (catch)





Try block-: try block contains code which may cause exception.

Syntax-:

try:

exception

except-: the except block is use to catch the exception that is raised in the try block. There can be multiple except block for try block.

Syntax-:

except ExceptionName:

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Output-:

```
Variable x is not defined
>>>
```

else-: this block is executed if no exception is raised. Else block is executed after try block.

Note-: if try block contains some exception then except block executes otherwise else block will execute.

Syntax-:

else:

    statements

Ex-: take a look how else block executes

Sol).

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Output-:

```
Hello
Nothing went wrong
>>>
```

Note-: since try block did not contain any exception so try and else both blocks executed.

finally-: this block will execute irrespective of whether there is an exception or not.

Syntax-:

finally

statements

Ex-:

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Outout-:

```
Something went wrong
The 'try except' is finished
>>>
```

Note-: we can have multiple except blocks to handle multiple exception.

Note-: we can write try block without any except block.

Ex-:

```
#The try block will raise an error when trying to write to a read-only file:

try:
  f = open("C:/Users/avira/Documents/python/text.txt")
  f.write("hello there")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
```

Output-:

```
Something went wrong when writing to the file
>>>
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

```
x = int(input("enter any number:"))
if x < 0:
  raise Exception("Sorry, no numbers below zero")
else:
    print(x)
```

Output-:

```
enter any number:-6
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\new.py", line 5, in <module
    raise Exception("Sorry, no numbers below zero")
Exception: Sorry, no numbers below zero
>>>
```

Ex-: You can define what kind of error to raise, and the text to print to the user.

```
x = "hello"
if not type(x) is int:
    raise TypeError("Only integers are allowed")
```

Output-:

```
Traceback (most recent call last):
  File "C:\Users\avira\Desktop\new.py", line 5, in <module>
    raise TypeError("Only integers are allowed")
TypeError: Only integers are allowed
>>>
```

Ex-: a program that shows how you can handle the raise exception

```
try:
    x=int(input('Enter a number upto 100: '))
    if x > 100:
        raise ValueError(x)
except ValueError:
    print(x, "is out of allowed range")
else:
    print(x, "is within the allowed range")
```

Output-:

```
Enter a number upto 100: 144
144 is out of allowed range
>>>
```

Ex). program to handle multiple errors with one except block.

Sol).

```
try :
    a = 3
    if a < 4 :
        # throws ZeroDivisionError for a = 3
        b = a/(a-3)

    # throws NameError if a >= 4
    print("Value of b = ", b)

# note that braces () are necessary here for multiple exceptions
except(ZeroDivisionError, NameError):
    print("\nError Occurred and Handled")
```

Output-:

```
Error Occurred and Handled
>>>
```

Ex-: a function program that will check for errors and behave accordingly

Sol).

```
def myfun(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print("a/b result in 0")
    else:
        print(c)

# Driver program to test above function
myfun(2.0, 3.0)
myfun(3.0, 3.0)
```

Output-:

```
-5.0
a/b result in 0
>>>
```