Unit 1:

Beginning with Python, Datatypes, Operators, I/O and Control statements

Que 1 : What is Python

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.
- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Que 2 : Python Features

1) Easy to Learn and Use

- Python is easy to learn and use. It is developer-friendly and high level programming language.

2) Expressive Language

- Python can perform complex tasks using a few lines of code.
- A simple example, the hello world program you simply type **print("Hello World")**.
- It will take only one line to execute, while Java or C takes multiple lines.

3) Interpreted Language

- Python is an interpreted language i.e. interpreter executes the code line by line at a time.
- This makes debugging easy and thus suitable for beginners..

4) Cross-platform Language

- Python can run equally on different platforms such as Windows, Linux, Unix and Macintosh etc. So, we can say that Python is a portable language.

5) Free and Open Source

- Python language is freely available at official web address.
- The source-code is also available. Therefore it is open source.

6) Object-Oriented Language

- Python supports object-oriented language and concepts of classes and objects come into existence.

7) Extensible

- It implies that other languages such as C/C++ can be used to compile the code and thus it can be used further in our Python code.

8) Large Standard Library

- Python has a large and broad library and provides rich set of module and functions for rapid application development.

9) GUI Programming Support

- Graphical user interfaces can be developed using Python.

10) Integrated

- It can be easily integrated with languages like C, C++, JAVA etc.

Que 3 : Flavours of Python

- **Flavors** of Python simply refers to the different Python compilers.

1.CPython :

- CPython is the Python compiler implemented in C programming language.
- In this, Python code is internally converted into the **byte code** using standard C functions. Additionally, it is possible to run and execute programs written in C/C++ using CPython compiler

2. Jython :

- Earlier known as JPython.
- Jython is an implementation of the Python programming language designed to run on the Java platform.
- Jython is extremely useful because it provides the productivity features of a mature scripting language while running on a JVM.

3. **PyPy** :

- This is the implementation using Python language.
- PyPy often runs faster than CPython because PyPy is a just-in-time compiler while CPython is an interpreter.

4. **IronPython** :

- IronPython is an open-source implementation of the Python programming language which is tightly integrated with the .NET Framework.

5. **RubyPython** :

- RubyPython is a bridge between the Ruby and Python interpreters.

- It embeds a Python interpreter in the Ruby application's process using FFI (Foreign Function Interface).

6. Pythonxy :

- Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python.

7. StacklessPython :

- Stackless Python is a Python programming language interpreter.
- In practice, Stackless Python uses the C stack, but the stack is cleared between function calls

8. AnacondaPython :

- Anaconda is a free and open-source distribution of the Python and Rprogramming languages for scientific computing, that aims to simplify package management and deployment.
- Package versions are managed by the package management system conda.

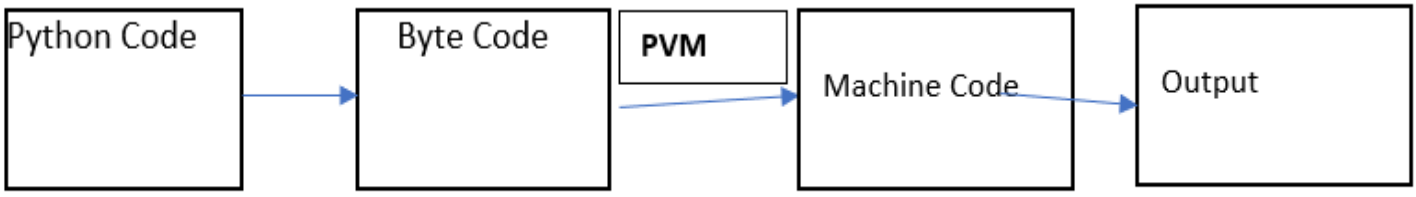
Que 4 : Explain PVM (Python Virtual Machine)or PBC(Python Byte Code) with Figure.

- As a programmer, we all know that a computer only understands machine language and every programming langu
-
- age converts its code to machine language.
- This is done by a compiler of that language.
- The Python compiler also does the same thing but in a slightly different manner.

When we run a Python program, two steps happen,
1. The code gets converted to another representation called 'Byte Code'
2. 'Byte Code' gets converted to Machine Code (which is understandable by the computer)

The second step is being done by PVM or Python Virtual Memory. So PVM is nothing but a software/interpreter that converts the byte code to machine code for given operating system.

PVM is also called Python Interpreter and this is the reason Python is called an Interpreted language.



Que 5 : Explain Python Memory Management

- Understanding Memory allocation is important to any software developer as writing efficient code means writing a memory-efficient code.
- Memory allocation can be defined as allocating a block of space in the computer memory to a program.
- In Python memory allocation and deallocation method is automatic as the Python developers created a garbage collector for Python so that the user does not have to do manual garbage collection.

1. Garbage Collection

- Garbage collection is a process in which the interpreter frees up the memory when not in use to make it available for other objects.
- Assume a case where no reference is pointing to an object in memory i.e. it is not in use so, the virtual machine has a garbage collector that automatically deletes that object from the heap memory

2. Reference Counting

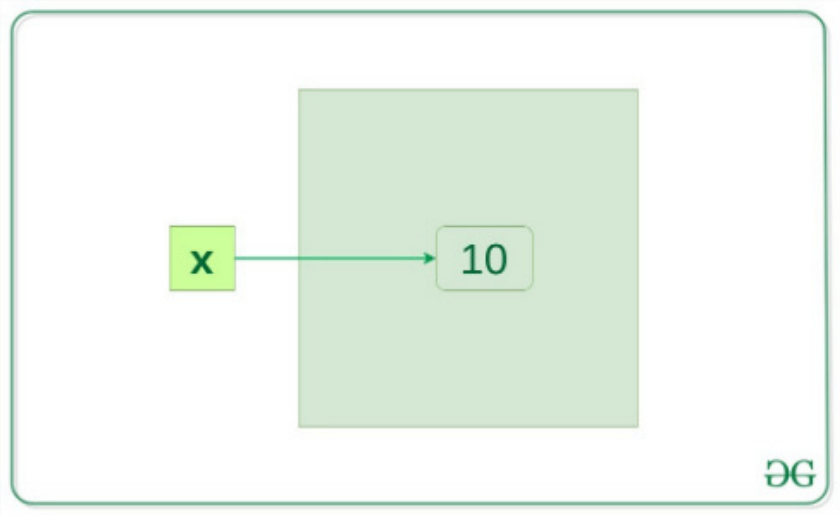
- Reference counting works by counting the number of times an object is referenced by other objects in the system.
- When references to an object are removed, the reference count for an object is decremented.
- When the reference count becomes zero, the object is deallocated.

For example, Let's suppose there are two or more variables that have the same value, so, what Python virtual machine does is, rather than creating another object of the same value in the private heap, it actually makes the second variable point to that originally existing value in the private heap. Therefore, in the case of classes, having a number of references may occupy a large amount of space in the memory, in such a case referencing counting is highly beneficial to preserve the memory to be available for other objects

Example:

```
x =10
```

When x = 10 is executed an integer object 10 is created in memory and its reference is assigned to variable x, this is because everything is object in Python.



Let’s verify if it’s true

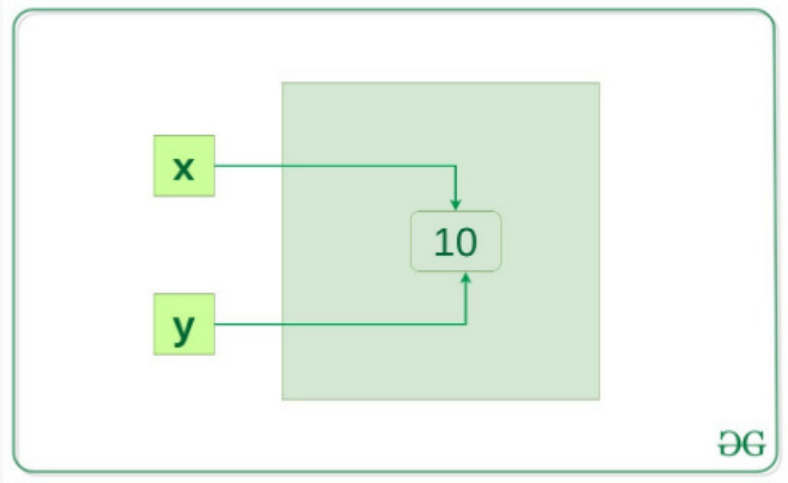
```
x =10
y =x

ifid(x) ==id(y):
   print("x and y refer to the same object")
```

Output:

x and y refer to the same object

In the above example, y = x will create another reference variable y which will refer to the same object because Python optimizes memory utilization by allocation the same object reference to a new variable if the object already exists with the same value.



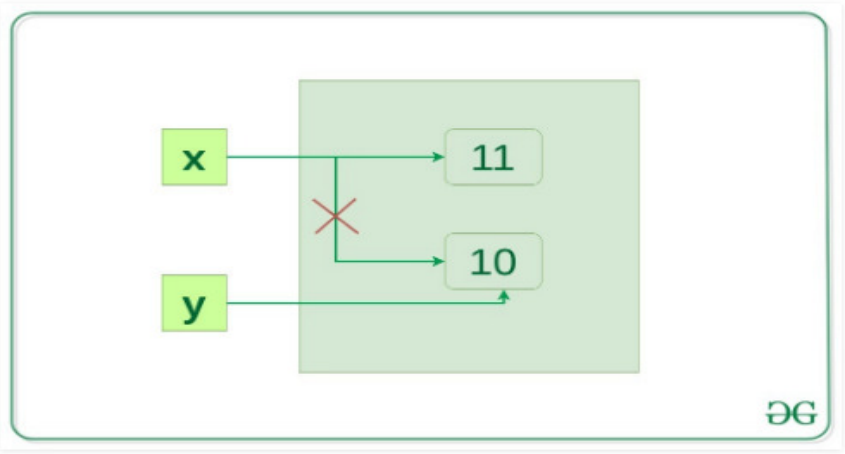
Now, let’s change the value of x and see what happens.

```
x =10
y =x
x +=1

ifid(x) !=id(y):
   print("x and y do not refer to the same object")
```

Output:

x and y do not refer to the same object
So now x refer to a new object x and the link between x and 10 disconnected but y still refer to 10.



3. Memory Allocation in Python

- There are two parts of memory:
  - stack memory
  - heap memory
  - The methods/method calls and the references are stored in **stack memory** and all the values objects are stored in a **private heap**.

4. Work of Stack Memory

- The allocation happens on nearby blocks of memory.
- We call it stack memory allocation because the allocation happens in the function call stack.
- The size of memory to be allocated is known to the compiler and whenever a function is called, its variables get memory allocated on the stack.
- It is the memory that is only needed inside a particular function or method call. When a function is called, it is added onto the program's call stack.
- Any local memory assignments such as variable initializations inside the particular functions are stored temporarily on the function call stack, where it is deleted once the function returns, and the call stack moves on to the next task.
- This allocation onto a contiguous block of memory is handled by the compiler using predefined routines, and developers do not need to worry about it.

Example:

```
deffunc():

    # All these variables get memory
    # allocated on stack
    a =20
    b =[]
    c =""
```

5. Work of Heap Memory

- The memory is allocated during the execution of instructions written by programmers.

- Note that the name heap has nothing to do with the heap data structure. It is called heap because it is a pile of memory space available to programmers to allocated and de-allocate.
- The variables are needed outside of method or function calls or are shared within multiple functions globally are stored in Heap memory.

Example:

```
# This memory for 10 integers
# is allocated on heap.
a =[0]*10
```

Que 6:Explain Garbage Collection in Python.

- Python's memory allocation and deallocation method is automatic.
- The user does not have to preallocate or deallocate memory similar to using dynamic memory allocation in languages such as C or C++.

Python uses two strategies for memory allocation:

- Reference counting
- Garbage collection

- In Python version 2.0, the Python interpreter only used reference counting for memory management.
- Reference counting works by counting the number of times an object is referenced by other objects in the system.
- When references to an object are removed, the reference count for an object is decremented.
- When the reference count becomes zero, the object is deallocated. Ex-

```
# Literal 9 is an object
b = 9
# Reference count of object 9
# becomes 0.
b = 4
```

- The literal value 9 is an object.
- The reference count of object 9 is incremented to 1 in line 1.
- In line 2 its reference count becomes zero as it is dereferenced. So garbage collector deallocates the object.

Manual Garbage Collection

- Invoking the garbage collector manually during the execution of a program can be a good idea on how to handle memory being consumed by reference cycles.
- The garbage collection can be invoked manually in the following way:

```
# Importing gc module
import gc
# Returns the number of
# objects it has collected
# and deallocated
collected = gc.collect()
```

```
# Prints Garbage collector
# as 0 object
print("Garbage collector: collected","%d objects." % collected)
```

- There are two ways for performing manual garbage collection:
- time-based and event-based garbage collection.
- **Time-basedgarbage collection** is simple: the garbage collector is called after a fixed time interval.
- **Event-basedgarbage collection** calls the garbage collector on event occurrence. For example, when a user exits the application or when the application enters into idle state.

Que 7 :Python Naming Conventions

Packages

- Package names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

Modules

- Module names should be all lower case
- When multiple words are needed, an underscore should separate them
- It is usually preferable to stick to 1 word names

Classes

- Class names should follow the UpperCaseCamelCase convention
- Python's built-in classes, however are typically lowercase words
- Exception classes should end in "Error"

Global (module-level) Variables

- Global variables should be all lowercase
- Words in a global variable name should be separated by an underscore

Instance Variables

- Instance variable names should be all lower case
- Words in an instance variable name should be separated by an underscore
- Non-public instance variables should begin with a single underscore
- If an instance name needs to be mangled, two underscores may begin its name

Methods

- Method names should be all lower case
- Words in an method name should be separated by an underscore
- Non-public method should begin with a single underscore
- If a method name needs to be mangled, two underscores may begin its name

Method Arguments

- Instance methods should have their first argument named ‘self’.
- Class methods should have their first argument named ‘cls’

Functions

- Function names should be all lower case
- Words in a function name should be separated by an underscore

Constants

- Constant names must be fully capitalized
- Words in a constant name should be separated by an underscore

Que 8 :Difference between c++ JAVA and Python

C++	JAVA	PYTHON
Compiled Programming language	Compiled Programming Language	Interpreted Programming Language
Supports Operator overloading	Does not support Operator Overloading	Supports Operator overloading
Provide both single and multiple inheritance	Provide partial multiple inheritance using interfaces	Provide both single and multiple inheritance
Platform dependent	Platform Independent	Platform Independent
Does Not support threads	Has in build multithreading support	Supports multithreading
Has limited number of library support	Has library support for many concepts like UI	Has a huge set of libraries that make it fit for AI, datascience, etc.
Code length is a bit lesser, 1.5 times less that java.	Java has quite huge code.	Smaller code length, 3-4 times less than java.
Functions and variables are used	Every bit of code is inside a class.	Functions and variables can be

outside the class		declared and used outside the class also.
C++ program is a fast compiling programming language.	Java Program compiler a bit slower than C++	Due to the use of interpreter execution is slower.
Strictly uses syntax norms	Strictly uses syntax norms	Use of ; is not compulsory.
like ; and {}.	like punctuations , ; .	

Que 9 : Python Data Types

- Python Data Types are used to define the type of a variable.
- It defines what type of data we are going to store in a variable.
- The data stored in memory can be of many types.
- For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Built-in datatypes

- Python has various built-in data types which are as follow
- **Numeric** - int, float, complex
- **String** - str
- **Sequence** - list, tuple, range
- **Binary** - bytes, bytearray, memoryview
- **Mapping** - dict
- **Boolean** - bool
- **Set** - set, frozenset
- **None**– NoneType

Python Numeric Data Type

Python numeric data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1
var2 = 10
var3 = 10.023
```

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

Here are some examples of numbers –

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Example

Following is an example to show the usage of Integer, Float and Complex numbers:

```
# integer variable.
a=100
print("The type of variable having value", a, " is ", type(a))

# float variable.
b=20.345
print("The type of variable having value", b, " is ", type(b))

# complex variable.
c=10+3j
print("The type of variable having value", c, " is ", type(c))
```

Python String Data Type

- The string can be defined as the sequence of characters represented in the quotation marks.
- In Python, we can use single, double, or triple quotes to define a string.
- String handling in Python is a straightforward task since Python provides built-in functions and operators to perform operations in the string.
- In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.
- The operator * is known as a duplication operator as the operation "Python" *2 returns 'Python Python'.

- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

Example

```
str = 'Hello World!'
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str + "TEST") # Prints concatenated string
```

Output

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python List Data Type

- Python Lists are the most flexible compound data types.
- A Python list contains items separated by commas and enclosed within square brackets ([]).
- To some extent, Python lists are similar to arrays in C.
- One difference between them is that all the items belonging to a Python list can be of different data type where as C array can store elements related to a particular data type.
- The values stored in a Python list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example –

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)          # Prints complete list
print (list[0])       # Prints first element of the list
print (list[1:3])     # Prints elements starting from 2nd till 3rd
print (list[2:])      # Prints elements starting from 3rd element
print (tinylist * 2)  # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Python Tuple Data Type

- Python tuple is another sequence data type that is similar to a list.
- A Python tuple consists of a number of values separated by commas.
- Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.

For example –

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')

print (tuple)             # Prints the complete tuple
print (tuple[0])          # Prints first element of the tuple
print (tuple[1:3])        # Prints elements of the tuple starting from 2nd till 3rd
print (tuple[2:])         # Prints elements of the tuple starting from 3rd element
print (tinytuple * 2)     # Prints the contents of the tuple twice
print (tuple + tinytuple) # Prints concatenated tuples
```

This produce the following result –

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists –

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
```

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

Python Ranges

- Python range() is an in-built function in Python which returns a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.
- We use range() function with for and while loop to generate a sequence of numbers.
- Following is the syntax of the function:

```
range(start, stop, step)
```

Here is the description of the parameters used:

- start: Integer number to specify starting position, (Its optional, Default: 0)
- stop: Integer number to specify starting position (It's mandatory)
- step: Integer number to specify increment, (Its optional, Default: 1)

Examples

Following is a program which uses for loop to print number from 0 to 4 –

```
for i in range(5):
  print(i)
```

This produce the following result –

```
0
1
2
3
4
```

Now let's modify above program to print the number starting from 1 instead of 0:

```
for i in range(1, 5):
  print(i)
```

This produce the following result –

```
1
2
3
  4
```

Again, let's modify the program to print the number starting from 1 but with an increment of 2 instead of 1:

```
for i in range(1, 5, 2):
  print(i)
```

This produce the following result −

```
1
3
```

PythonDictionary

- Python dictionaries are kind of hash table type.
- They work like associative arrays or hashes found in Perl and consist of key-value pairs.
- A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- For example −

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
print (dict['one'])       # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

This produce the following result −

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

- Python dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

Python Boolean Data Types

- Python boolean type is one of built-in data types which represents one of the two values either True or False.

- Python bool() function allows you to evaluate the value of any expression and returns either True or False based on the expression.

Examples

Following is a program which prints the value of boolean variables a and b –

```
a = True
# display the value of a
print(a)
# display the data type of a
print(type(a))
```

This produce the following result –

```
true
<class 'bool'>
```

Following is another program which evaluates the expressions and prints the return values:

```
# Returns false as a is not equal to b
a = 2
b = 4
print(bool(a==b))
# Following also prints the same
print(a==b)
# Returns False as a is None
a = None

print(bool(a))
# Returns false as a is an empty sequence
a = ()
print(bool(a))
# Returns false as a is 0
a = 0.0
print(bool(a))
# Returns false as a is 10
a = 10
print(bool(a))
```

This produce the following result –

```
False
False
```

False
False
False
True

Python Set

- Sets are used to store multiple items in a single variable.
- Set items are unchangeable, but you can remove items and add new items
- Set has only unique value

Example

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

# Note: the set list is unordered, meaning: the items will appear in a random order.

# Refresh this page to see the change in the result.
```

Python frozenset

- The frozenset() function returns an unchangeable frozenset object (which is like a set object, only unchangeable).

```
mylist = ['apple', 'banana', 'cherry']
x=frozenset(mylist)
x[1]="ihina"
print(x)
```

Bytes Sequences

- The Bytes daya type is use to represent a group of byte number
- Bytes is just like an array which can hold multiple value
- Bytes can store number between 0 to 255
- Bytes can not hold negative numbers

Example

```
a=["shree umiya bca college"]
b=bytes(b'a')
print(a)
```

Bytearray

- Bytearray is same like bytes but difference is only that you modify element in bytearray**.**
  Example
  #read an element form bytearray

```
x=bytearray(b'umiya college')
#delete an element from bytearray
del x[2:5]
print(x)
#modify en element from an byte array
x[2:5]=b"iya  "
print(x)
```

Que 10: How to convert one data type to another data type explain with example.

Or

Write a short note on Type Casting

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.
- Python has two types of type conversion.
  - Implicit Conversion
  - Explicit conversion

Implicit Conversion

- In Implicit type conversion, Python automatically converts one data type to another data type.
- This process doesn't need any user involvement.
- Let's see an example where Python promotes conversion of lower data type (integer) to higher data type (float) to avoid data loss.

  Example 1

  Converting integer to float

```
num_int = 123
num_flo = 1.23
num_new = num_int + num_flo
print("datatype of num_int:",type(num_int))
print("datatype of num_flo:",type(num_flo))
 print("Value of num_new:",num_new)
print("datatype of num_new:",type(num_new))
```

- When we run the above program, the output will be datatype of num_int: <class 'int'> datatype of num_flo: <class 'float'> Value of num_new: 124.23 datatype of num_new: <class 'float'>

- In theabove program
  - We add two variables num_int and num_flo, storing the value in num_new.
  - We will look at the data type of all three objects respectively.
  - In the output we can see the datatype of num_int is an integer, datatype of num_flo is a float.
  - Also, we can see the num_new has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.

Explicit Type Conversion:

- In Explicit Type Conversion, users convert the data type of an object to required data type.
- We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.
- This type conversion is also called typecasting because the user casts (change) the data type of the objects.
  Syntax :
  (required_datatype)(expression)
- Typecasting can be done by assigning the required data type function to the expression.
  Example : Addition of string and integer using explicit conversion

```
a=23
b="23"
print("Data type of a=",type(a))
print("Data type of b=",type(b))
c=int(b)
print("Data type of c after conversion=",type(c))
d=a+c
print(c)
```

- In above program,
  - We add a and b variable.
  - We converted b from string(higher) to integer(lower) type using int() function to perform the addition.
  - After converting b to a integer value Python is able to add these two variable.
  - We got the d value and data type to be integer.

Que 11:Write a short note on IDLE Window.

- IDLE is Python's Integrated Development and Learning Environment.
- IDLE is integrated development environment (IDE) for editing and running Python 2.x or Python 3 programs.
- The IDLE GUI (graphical user interface) is automatically installed with the Python interpreter.
- IDLE was designed specifically for use with Python.
- IDLE has a number of features to help you develop your Python programs including powerful syntax highlighting.
  IDLE provides
- a text editor with syntax highlighting, auto completion, and smart indentation,
- a shell with syntax highlighting, and
- an integrated debugger, which you should ignore for now.
  - When IDLE starts it will open a shell window into which you can type Python commands.
  - It will also provide you with a file menu and an edit menu (as well as some other menus, which you can safely ignore for now).
- The file menu includes commands to

- create a new editing window into which you can type a Python program, open a file containing an existing Python program, and
- save the contents of the current editing window into a file (with file extension .py).
- The edit menu includes standard text-editing commands (e.g., copy, paste, and find) plus some commands specifically designed to make it easy to edit Python code (e.g., indent region and comment out region)

Que 12. Explain Python keywords & identifiers

Python Keywords

- Keywords are the reserved words in Python.
- 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.
- In Python, keywords are case sensitive.
- There are 33 keywords in Python 3.7.
- This number can vary slightly in the course of time.
- All the keywords except True, False and none are in lowercase and they must be written as it is. The list of all the keywords is given below.

Python Identifiers.

- An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Rules for writing identifiers

- Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _. Names like myClass, var_1 and

s in Python				
False	class	finally	is	return
None	continue	for	lambda	Try
True	def	from	nonlocal	While
and	del	global	not	With
As	elif	if	or	Yield
assert	else	import	pass	
break	except	in	raise	

print_this_to_screen, all are valid example.

- An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
- Keywords cannot be used as identifiers.

```
>>> global = 1
File "<interactive input>", line 1
global = 1
^
SyntaxError: invalid syntax
```

We cannot use special symbols like !, @, #, $, % etc. in our identifier.

```
>>> a@ = 0
File "<interactive input>", line 1
a@ = 0
^
SyntaxError: invalid syntax
```

- Identifier can be of any length.
- Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

Que 13.Explain Python mapping Types.

- The mapping objects are used to map hash table values to arbitrary objects.
- In python there is mapping type called dictionary.
- It is mutable. mutable means k hmne jo create kiya hai usme changes kr skte hai
- The keys of the dictionary are arbitrary.
- As the value, we can use different kind of elements like lists, integers or any other mutable type objects.
- Some dictionary related methods and operations are –

Method len(d)

- The len() method returns the number of elements in the dictionary.

```
dist={1:"heena",2:"miral",3:"komal"}
print(len(dist))
```

Operation d[k]

- It will return the item of d with the key ‘k’. It may raise KeyError if the key is not mapped.

```
dist={1:"heena",2:"miral",3:"komal"}
print(dist.keys(2))
```

Method get(key[, default])

- The get() method will return the value from the key. The second argument is optional. If the key is not present, it will return the default value.

```
dist={1:"heena",2:"miral",3:"komal"}
print(dist.get(2))
print(dist.get(4,"tulsi"))
```

Method items()

- It will return the items using (key, value) pairs format.
  dist={1:"heena",2:"miral",3:"komal"}
  print(dist.items())

Method keys()

- Return the list of different keys in the dictionary.
  dist={1:"heena",2:"miral",3:"komal"}
  print(dist.keys())

Method values()

- Return the list of different values from the dictionary.
  dist={1:"heena",2:"miral",3:"komal"}
  print(dist.values())

Method update(elem)

- Modify the element elem in the dictionary.

  dist={1:"heena",2:"miral",3:"komal"}
  dist.update({2:"dixita"})
  print(dist)

Que 14.Operators, I/O and control statements

Python Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Logical operators
  - Identity operators
  - Membership operators
  - Bitwise operators

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y

*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3

%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y

<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

- The identity operators in Python are used to determine whether a value is of a certain class or type.
- They are usually used to determine the type of data a certain variable contains. For example, you can combine the identity operators with the built-in type() function to ensure that you are working with the specific variable type.
- Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y

is not	Returns True if both variables are not the same object	x is not y

Example

x=2

y=3

print(x is y)

print(x is not y)

Python Membership Operators

- These operators test whether a value is a member of a sequence.
- The sequence may be a list, a string, or a tuple.
- These operator returns either True or False, if a value/variable found in the list, its returns Trueotherwise it returns False.
- We have two membership python operators- 'in' and 'not in'.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Example

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

print("banana" in x)

# returns True because a sequence with the value "banana" is in the list
```

Example2

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

print("banana" not in x)
```

Que 15.Output statements, Input statements

- In this article, we are going to discuss some two basic yet important input and output functions and their variants provided by the Python.

- The input function is used to takean input from the user at the console, while

Parameters	Description
value	This is a single object value passed to the print() function, which will be printed.
...	The three dots **...** signifies that we can even pass multiple objects(separated by commas) to the print() function.
sep	This is a separator used to separate the objects passed to the print() function. Default value of **sep** is ' ' i.e. a space.
end	The value of **end** parameteris printed at the last i.e. after all the objects passed to print() function, are printed. Default value of **end** is '\n' i.e. a newline character.
file	If the value of **file** is specified then the print() function can be used to print the objects in the file, otherwise, by default **sys.stdout** prints the objects on the console.
flush	The **flush** can only be given a bool value. By default, the value of flush is **False**, if **True** the output stream is flushed.

the output function will generate an output at the console.

- So, let us see what these important input/output functions are –

Functions	Description
print()	This output function, used to generate an output at the console.
input()	1) This input function is used to read a line of input entered by the user at the console and returns it as a string.

- **Using the print() output function**

- In Python, the **print** function is the first and the most basic output function one can use.
- It is used to generate an output at the console.
  Syntax of **print()** function -
  print(value, ..., sep = ' ', end = '\n' , file = sys.stdout, flush = False)

Parameters	Description
value	This is a single object value passed to the print() function, which will be printed.
...	The three dots **...** signifies that we can even pass multiple objects(separated by commas) to the print() function.
sep	This is a separator used to separate the objects passed to the print() function. Default value of **sep** is ' ' i.e. a space.
end	The value of **end** parameteris printed at the last i.e. after all the objects passed to print() function, are printed. Default value of **end** is '\n' i.e. a newline character.
file	If the value of **file** is specified then the print() function can be used to print the objects in the file, otherwise, by default **sys.stdout** prints the objects on the console.
flush	The **flush** can only be given a bool value. By default, the value of flush is **False**, if **True** the output stream is flushed.

Example of print() function

```
# Python - The print() function

i = 10
f = 10.8
d = 19.7907
str1 = 'Hello'
```

```
# Passing a single object to be printed to the print() function
print('Hello World!')


# Passing multiple objects to be printed to the print() function
print(i, f, d, str1)


# Passing multiple objects to be printed to the print() function
# And, also passing it arguments to alter with its default behaviour
print(i, f, d, str1, sep='|', end='End')
```

Output

```
Hello World!
10 10.8 19.7907 Hello
10|10.8|19.7907|HelloEnd
```

Using the print() output function to write to a file on disk

Using this function, we can fill the unused white spaces in a value(to be printed at the console), with a character of our choice Let's see the syntax of **fill()** function -

```
file1=open('10.txt',"W")

print("hello",file=file1,flush=True)

file1.close()
```

Output

Executing this program will create a new file named **File1.txt** in your current directory and will write the value of objects which were passed to the **print()** function. The file with its content looks like :

hello

- **Using the input() function**

The **input** function is used to take the input entered by the user at the console and returns the input as a string

Syntax of **input** function -

```
input(prompt=None)
```

Where, the **prompt** is passed a string value, which will be act a prompt to the user to enter a value. Default value of **prompt** is **None.**

Example of input() function -

```
str1=input("enter your name")
print(str1)
str=input()
print(str)
```

As you can see in the first call to **input()** function, we have simply passed it string object, which will prompt the user with a message to enter an input.
In the second call to **input()** function, we have not passed any string to it i.e. a blank prompt to the user.

Integer,float input in Python

Python takes all the input as a string input by default. To convert it to any other data type we have to convert the input explicitly. For example, to convert the input to int or float we have to use the int() and float() method respectively.

```
# Taking input from the user as integer
num = int(input("Enter a number: "))
add = num + 1
# Output
print(add)
```

How to take Multiple Inputs in Python :

we can take multiple inputs of the same data type at a time in python, using map() method in python.

```
a,b,c=map(int,input("enter values").split())
print("The numbers are:",a,b,c)
d=a+b+c
print(d)
```

How take inputs for the Sequence Data Types like List, Set, Tuple, etc.

In the case of List and Set the input can be taken from the user in two ways.

1. Taking List/Set elements one by one by using the append()/add() methods.
2. Using map() and list() / set() methods.

Taking List/Set elements one by one

Take the elements of the List/Set one by one and use the append() method in the case of List, and add() method in the case of a Set, to add the elements to the List / Set.

```
List=list()
Set=set()`        ``````````````````````
l=int(input("Enter size of list to enter items in list"))
s=int(input("Enter size of set to insert items in set"))
print("enter value for list")
for i in range(0,l):
   List.append(int(input()))
print("enter value for list")
for i in range(0,s):
   Set.add(int(input()))
print(List)
print(Set)
```

OUTPUT

```
Enter size of list to enter items in list2
Enter size of set to insert items in set2
enter value for list
12
34
enter value for list
12
34
[12, 34]
{34, 12}
```

Using map() and list() / set() Methods

```
List=list(map(int,input("entr value").split()))
Set=set(map(int,input("enter value for set").split()))
print(List)
print(Set)
```

OUTPUT

```
entr value12 34 45
enter value for set12 34 56
```

[12, 34, 45]
{56, 34, 12}

Taking Input for Tuple

- We know that tuples are immutable, there are no methods available to add elements to tuples.
- To add a new element to a tuple, first type cast the tuple to the list, later append the element to the list, and again type cast list back to a tuple.

Example

```
T=(1,2,3,4,5)
print("tuple before adding new element")
print(T)
L=list(T)
L.append(int(input("Enter new element")))
T=tuple(L)
print("Tuple after adding the new element")
print(T)
```

Que 16: Explain Python Command Line Arguments

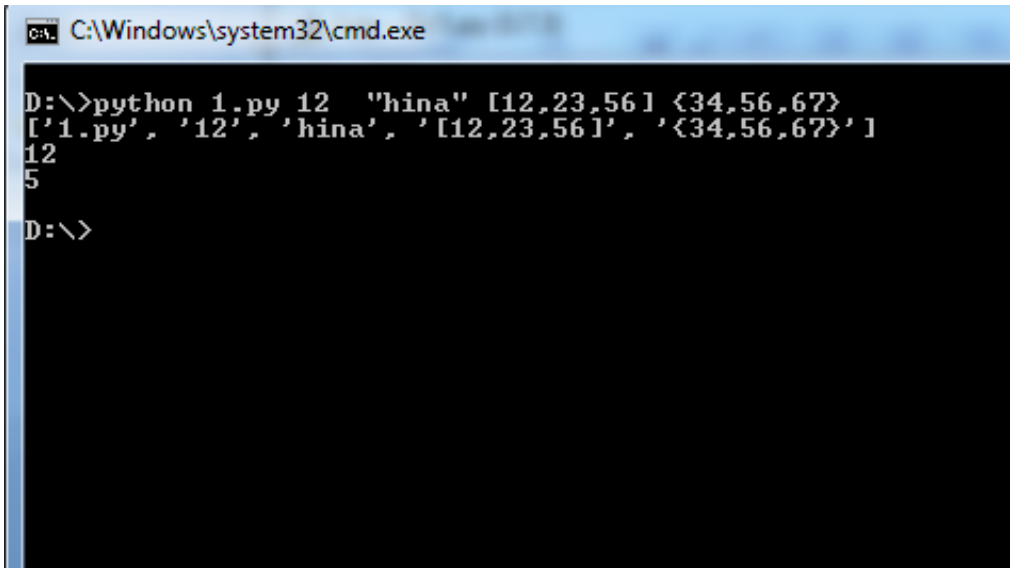
- Python Command Line Arguments provides a convenient way to accept some information at the command line while running the program.
- The arguments that are given after the name of the Python script are known as Command Line Arguments and they are used to pass some information to the program.

1. **Using sys.arg**
   - The sys module provides functions and variables used to manipulate different parts of the Python runtime environment.
   - This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
   - One such variable is sys.argv which is a simple list structure. It's main purpose are:
     - It is a list of command line arguments.
     - len(sys.argv) provides the number of command line arguments.
     - sys.argv[0] is the name of the current Python script.

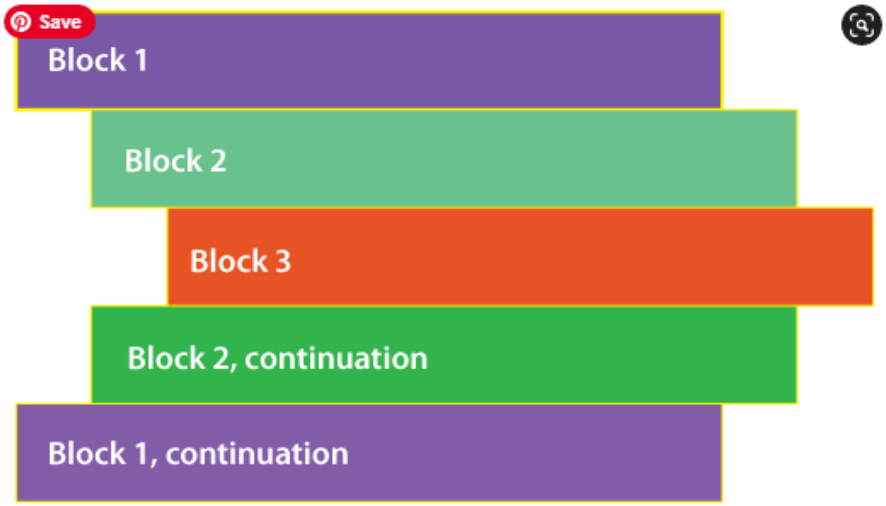
   Example

```
import sys
print(sys.argv)
print(sys.argv[1])
print(len(sys.argv))
```

Output



Que 17: Explain A word on Indentation in python



Advantages of Indentation in Python

- Indentation is used in python to represent a certain block of code, but in other programming languages, they refer to various brackets. Due to indentation, the code looks more efficient and beautifully structured.
- Indentation rules used in a python programming language are very simple; even a programmer wants their code to be readable.
- Also, indentation increases the efficiency and readability of the code by structuring it beautifully.

Disadvantages of Indentation in Python

- Due to the uses of whitespaces in indentation, sometimes it is a very tough task to fix the indentation error when there are many lines of code.
- The various popular programming languages like C, C++, Java use braces for indentation, so anybody coming from the other side of the developed world finds it hard to adjust to the idea of using whitespaces for the indentation.

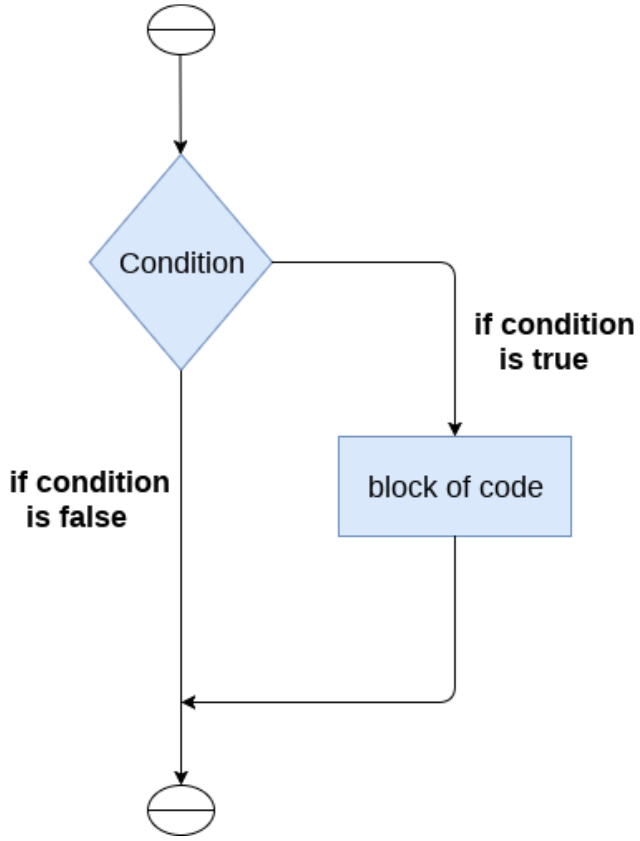
Que 18: Explain The If-else statements,if-elif-else statement

- decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.
- In python, decision making is performed by the following statements.

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

The if statement

- The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block.
- The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

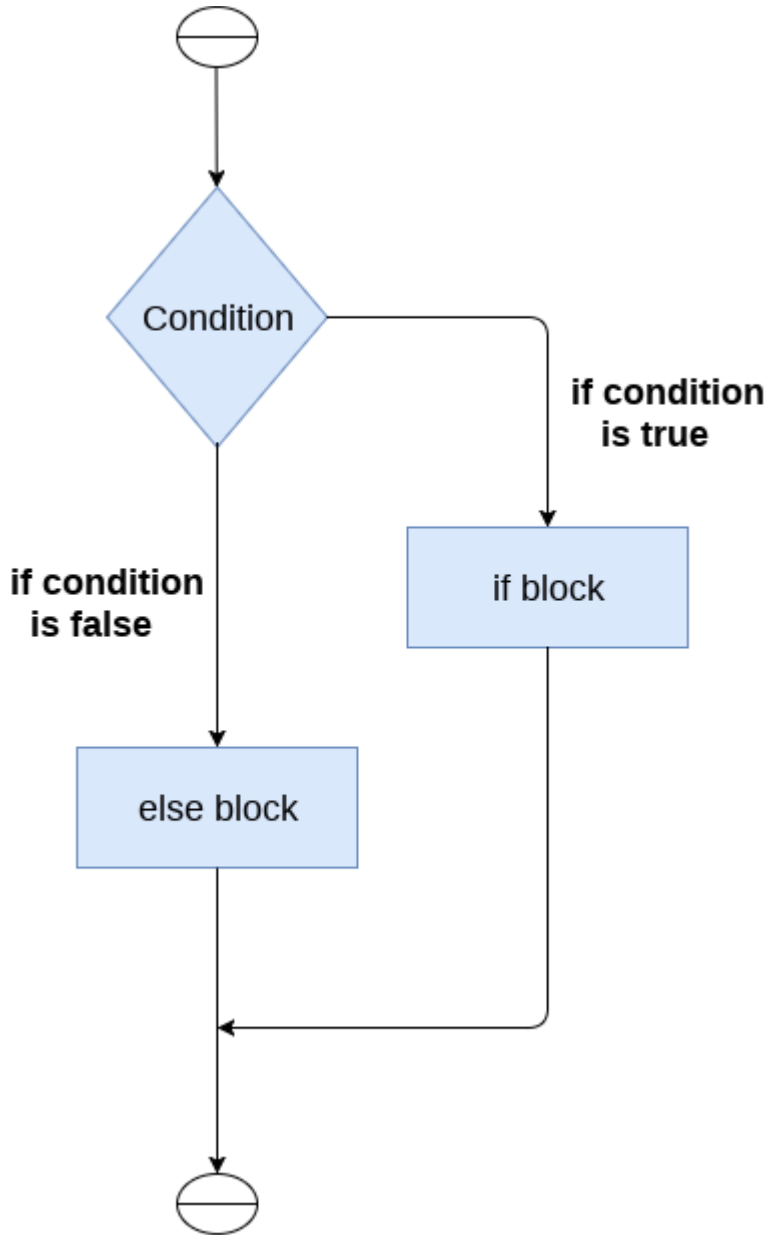
```
if expression:
   statement
```

Example

```
a=-45
if a>0:
        print("a is positive")
```

The if-else statement

- The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.
- If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



Syntax

```
if condition:
  #block of statements
else:
 #another block of statements (else-block)
```

Example

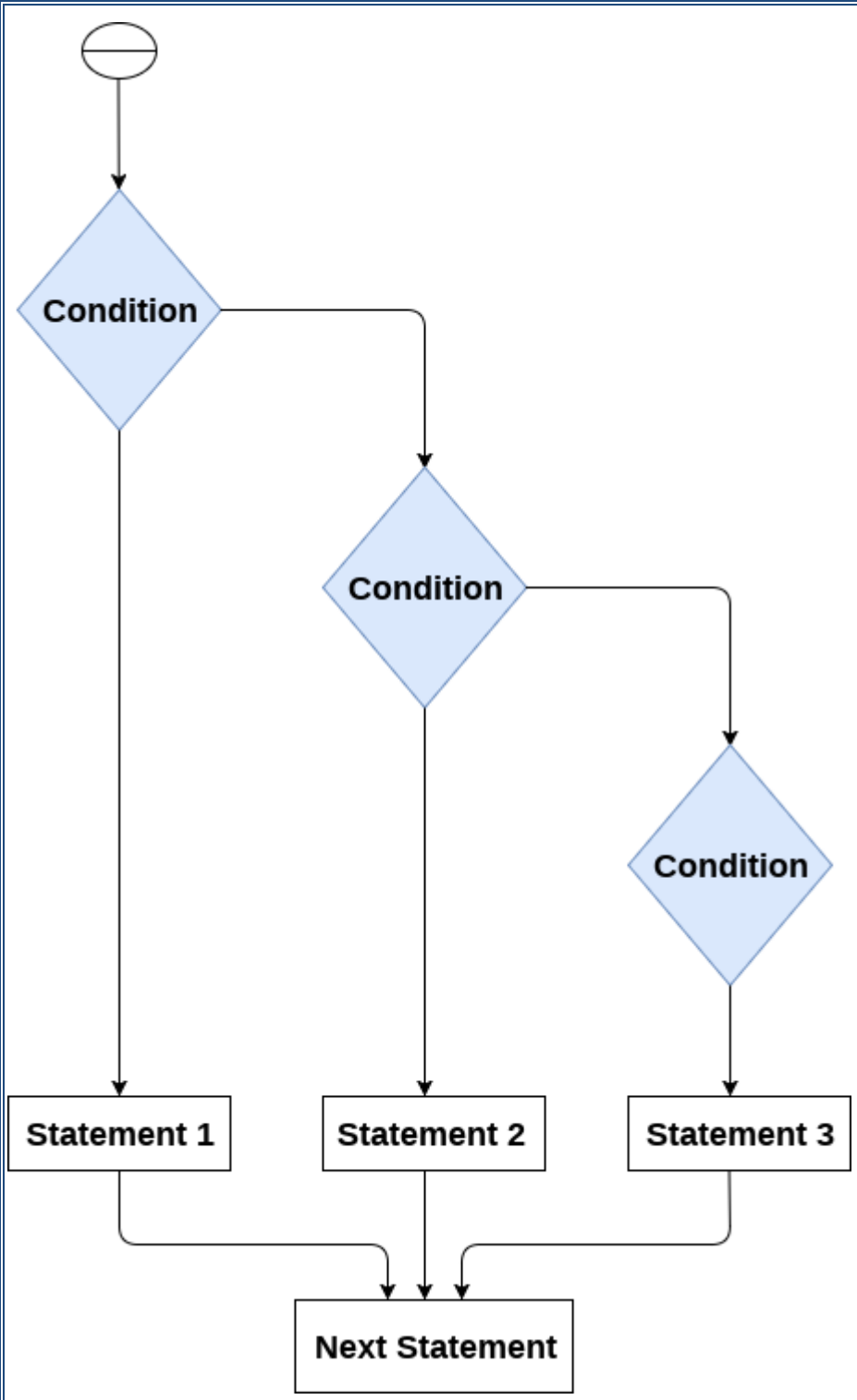
```
a=-45
if a>0:
        print("a is positive")
else:
   print("a is negeative")
```

The elif statement

- The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.
- We can have any number of elif statements in our program depending upon our need. However, using elif is optional.
- The elif statement works like an if-else-if ladder statement in C.
- It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements
elif expression 2:
   # block of statements
elif expression 3:
   # block of statements
else:
 # block of statements
```



Example

```
a=0
if a>0:
        print("a is positive")
elif a<0:
    print("a is negative")
else:
    print("a is zero")
```

nested if statement

You can have if statements inside if statements, this is called nested if statements.

Santax:

```
if (condition1):
  # Executes when condition1 is true
  if (condition2):
    # Executes when condition2 is true
  # if Block is end here
# if Block is end here
```

Example

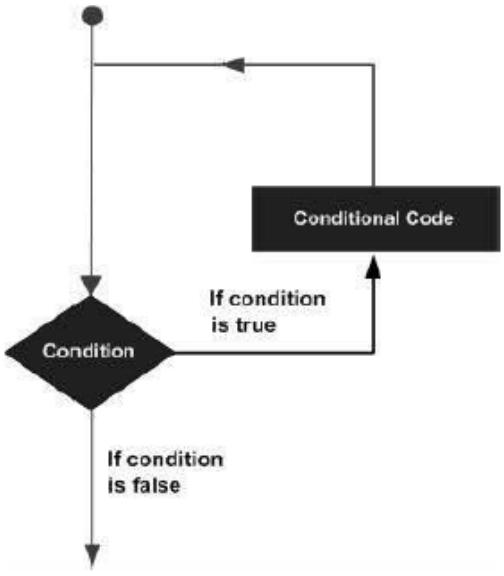
```
a=-10
if a<=0:

   if a==0:
      print(" a is zero")
   else:
      print(" a is negative")

else:
   print("a is positive")
```

Que 19:Explain types of loops in python

- A loop statement allows us to execute a statement or group of statements multiple times.
- The following diagram illustrates a loop statement –



- Python programming language provides following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	**while loop** Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	**for loop** Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	**nested loops** You can use one or more loop inside any another while, for or do..while loop.
4	**infinite loop** in Python is a continuous repetition of the conditional loop until some external factors like insufficient CPU memory

While loop Syntax

```
while condition:
   # body of while loop
```

Example

```
a=0
while (a<9):
   a=a+2
   print(a)
```

for loop Syntax

```
for val in sequence:
   # statement(s)
```

Example

```
for i in range(1,7,2):
    print(i)
```

Nested loop Example

```
colour = ["red", "blue", "green"]
fruits = ["apple", "banana", "cherry"]
for x in colour:
   for y in fruits:
      print(x, y)
```

The continue Statement

- With the continue statement we can stop the current iteration of the loop, and continue with the next:

Example

```
fruits = ["apple", "banana", "cherry"]
```

```
for x in fruits:
 if x == "banana":
  continue

 print(x)
```

The break Statement

- With the break statement we can stop the loop before it has looped through all the items:

Example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

The else Suite

- In most of the programming languages (C/C++, Java, etc), the use of else statement has been restricted with the if conditional statements.
- But Python also allows us to use the else condition with for loops.
    - **for with else syntax**

```
for( var in sequence)
statements
else: statements
```

Example:

```
for i in range(1,5):
      print("yes")
else:
      print("no")
```

Example 1:

```
group1 = [1,2,3,4,5]
search = int(input('Enter element to search:'))
for element in group1:
  if search == element:
      print('Element found in group')
      break #come out of for loop
else:
      print('Element not found in group1')
```

    - **while with else syntax**

```
while( condition ):
 statements
else:
statements
```

Example

```
i=0
while (i<5):
```

```
  i=i+1
  print("yes")
else:
      print("no")
```

The Pass Statement

- When the user does not know what code to write, So user simply places a pass at that line.
- Sometimes, the pass is used when the user doesn't want any code to execute.
- So users can simply place a pass where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements.
- So using a pass statement user avoids this error.

Example

```
a = 10
b = 20

if(a<b):
  pass
else:
  print("b<a")
```

The assert Statement

- In simpler terms, we can say that assertion is the boolean expression that checks if the statement is True or False.
- If the statement is true then it does nothing and continues the execution, but if the statement is False then it stops the execution of the program and throws an error.

Example

```
# initializing number
a = 4
b = 0

# using assert to check for 0
print("The value of a / b is : ")
assert b != 0
print(a / b)
```