1.Explain about Input and output Statements in Python with examples.

Every computer application should have a provision to accept input from the user when it is running. This makes the application interactive. Depending on how it is developed, an application may accept the user input in the form of text entered in the console **(sys.stdin)**, a graphical layout, or a web-based interface.

Python utilizes specific built-in functions for handling input and output operations, enabling interaction between a program and the user or external files.

Python provides us with two built-in functions to read the input from the keyboard.

- The input () Function
- The raw_input () Function

Python interpreter works in interactive and scripted mode. While the interactive mode is good for quick evaluations, it is less productive. For repeated execution of same set of instructions, scripted mode should be used.

Input Function:

The input() function is used to obtain data from the user via the console.

```
user_input = input("Enter your name: ")
print("Hello,", user_input)
```

- When input() is called, it displays the provided prompt string (if any) to the user.
- The program then pauses, awaiting user input, which is captured when the user presses Enter.
- The input() function always returns the received data as a string, regardless of whether the user enters numbers or other characters.
- If numerical input is required for calculations, type conversion functions like int() or float() must be used to convert the string to the appropriate numeric type.

Ex:-

```
age_str = input("Enter your age: ")
age_int = int(age_str)
print("You are", age_int, "years old.")
```

Take Multiple Input in Python

We are taking multiple input from the user in a single line, splitting the values entered by the user into separate variables for each value using the split() method. Then, it prints the values with corresponding labels, either two or three, based on the number of inputs provided by the user.

```
# taking two inputs at a time
x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)
```

The raw_input() Function

The **raw_input()** function works similar to **input()** function. Here only point is that this function was available in Python 2.7, and it has been renamed to **input()** in Python 3.6

Following is the syntax of the **raw_input()** function:

```
var = raw_input ([prompt text])
```

Ex:-

```
name = raw_input("Eneter your name - ")
city = raw_input("Enter city name - ")
print ("Hello My name is", name)
print ("I am from ", city)
```

When you run, you will find the cursor waiting for user's input. Enter values for name and city. Using the entered data, the output will be displayed.

The print() Function

Python's print() function is a built-in function. It is the most frequently used function, that displays value of Python expression given in parenthesis, on Python's console, or standard output

print(object= separator= end= file= flush=)

Here,

- **object** - value(s) to be printed
- **sep** (optional) - allows us to separate multiple **objects** inside print().
- **end** (optional) - allows us to add add specific values like new line "\n", tab "\t"
- **file** (optional) - where the values are printed. It's default value is sys.stdout (screen)
- **flush** (optional) - boolean specifying if the output is flushed or buffered. Default: False

Example 1: Python Print Statement

```
print('Good Morning!')

print('It is rainy today')
```

Example 2: Python print() with end Parameter

```
# print with end whitespace

print('Good Morning!', end= ' ')

print('It is rainy today')
```

above we have included the end= ' ' after the end of the first print() statement.Hence, we get the output in a single line separated by space.

Example 3: Python print() with sep parameter

```
print('New Year', 2023, 'See you soon!', sep= '. ')
```

Example: Print Python Variables and Literals

We can also use the print() function to print Python variables. For example,

Ex:-

```
number = -10.6
name = "Programiz"
# print literals
print(5)
# print variables
print(number)
print(name)
```

Q2. Discuss about Command Line Arguments in Python.

Command line arguments in Python are values passed to a script when it is executed from the terminal or command prompt. These arguments allow users to provide input to a program and modify its behavior without altering the source code.

There are two primary ways to handle command line arguments in Python:

Using sys.argv.

The sys module provides access to system-specific parameters and functions, including sys.argv.

This is a list containing all command line arguments, where sys.argv[0] is the script name itself, and subsequent elements are the arguments passed.

Example

```
import sys
print(f"Script name: {sys.argv[0]}")
if len(sys.argv) > 1:
    print(f"First argument: {sys.argv[1]}")
    print(f"All arguments (excluding script name): {sys.argv[1:]}")
```

run this script (e.g., my_script.py) with arguments:

Code

```
python my_script.py arg1 arg2 "argument with spaces"
```

Using the argparse module.

For more complex command-line interfaces, the argparse module is recommended. It simplifies the process of defining, parsing, and validating arguments, including handling options, positional arguments, and generating help messages.

Python

```
import argparse
parser = argparse.ArgumentParser(description="A sample script demonstrating
argparse.")
parser.add_argument("name", type=str, help="Your name")
parser.add_argument("--greeting", "-g", type=str, default="Hello", help="The greeting
message")
args = parser.parse_args()
print(f"{args.greeting}, {args.name}!")
```

To run this script (e.g., greet_script.py):

Code

```
python greet_script.py Alice
python greet_script.py Bob --greeting "Hi there"
python greet_script.py Charlie -g "Greetings"
```

Q3. Explain about controlling program with decision statements.

Python program control flow is regulated by various types of conditional statements, loops, and function calls. By default, the instructions in a computer program are executed in a sequential manner, from top to bottom, or from start to end. However, such sequentially executing programs can perform only simplistic tasks. We would like the program to have a decision-making ability, so that it performs different steps depending on different conditions.

Most programming languages including Python provide functionality to control the flow of execution of instructions. Normally, there are two type of control flow statements in any programming language and Python also supports them.

Decision Making Statements

Decision making statements are used in the Python programs to make them able to decide which of the alternative group of instructions to be executed, depending on value of a certain Boolean expression.

The if Statements

Python provides if..elif..else control statements as a part of decision marking. It consists of three different blocks, which are **if** block, **elif** (short of else if) block and **else** block.

If Conditional Statement

If statement is the simplest form of a conditional statement. It executes a block of code if the given condition is true

Syntax:

If condition:
    S1
    S2

Ex:-

```
age = 20
if age >= 18:
  print("Eligible to vote.")
```

If else Conditional Statement

If Else allows us to specify a block of code that will execute if the condition(s) associated with an if or elif statement evaluates to False. Else block provides a way to handle all other cases that don't meet the specified conditions.

Syntax:

If condtion:
    S1
    S2
else:
    S3
    S4

Example:

```
age = 10
if age <= 12:
  print("Travel for free.")
else:
  print("Pay for ticket.")
```

Short Hand if-else

The short-hand if-else statement allows us to write a single-line if-else statement.

```
marks = 45
res = "Pass" if marks >= 40 else "Fail"
print(f"Result: {res}")
```

elif Statement

elif statement in Python stands for "else if." It allows us to check multiple conditions, providing a way to execute different blocks of code based on which condition is true. Using elif statements makes our code more readable and efficient by eliminating the need for multiple nested if statements.

Ex:-

```
age = 25
if age <= 12:
  print("Child.")
elif age <= 19:
  print("Teenager.")
elif age <= 35:
  print("Young adult.")
else:
  print("Adult.")
```

Python Nested if Statement

A nested if statement in Python is an if statement located within another if or else clause. This nesting can continue with multiple layers, allowing programmers to evaluate multiple conditions sequentially. It's particularly useful in scenarios where multiple criteria need to be checked before taking an action.

Example of Nested If Statements

Let’s consider a practical example to understand how nested if statements work in Python.

```
age = 30
member = True

if age > 18:
  if member:
    print("Ticket price is $12.")
  else:
    print("Ticket price is $20.")
else:
  if member:
    print("Ticket price is $8.")
  else:
    print("Ticket price is $10.")
```

Python match-case Statement

A Python **match-case** statement takes an expression and compares its value to successive patterns given as one or more case blocks. Only the first pattern that matches gets executed. It is also possible to extract components (sequence elements or object attributes) from the value into variables.

With the release of Python 3.10, a pattern matching technique called **match-case** has been introduced, which is similar to the **switch-case** construct available in C/C++/Java etc.

Syntax:

```
match variable_name:
case 'pattern 1' : statement 1
case 'pattern 2' : statement 2
...
case 'pattern n' : statement n
```

Example:

```
def weekday(n):
  match n:
    case 0: return "Monday"
    case 1: return "Tuesday"
    case 2: return "Wednesday"
    case 3: return "Thursday"
    case 4: return "Friday"
    case 5: return "Saturday"
    case 6: return "Sunday"
    case _: return "Invalid day number"
print (weekday(3))
print (weekday(6))
print (weekday(7))
```

Q3. Explain about controlling program with Loop or repetitive statements.

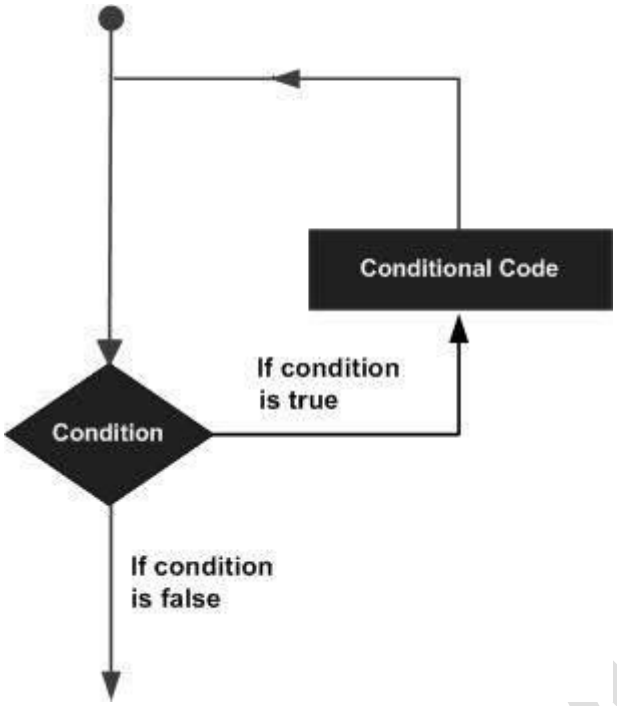
Python program control flow is regulated by various types of conditional statements, loops, and function calls. By default, the instructions in a computer program are executed in a sequential manner, from top to bottom, or from start to end. However, such sequentially executing programs can perform only simplistic tasks

Python loops allow us to execute a statement or group of statements multiple times.

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Flowchart of a Loop

The following diagram illustrates a loop statement –



Types of Loops in Python

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.

while Loop

A **while loop** in Python programming language repeatedly executes a target statement as long as the specified boolean expression is true.

This loop starts with **while keyword** followed by a boolean expression and colon symbol (:). Then, an indented block of statements starts.

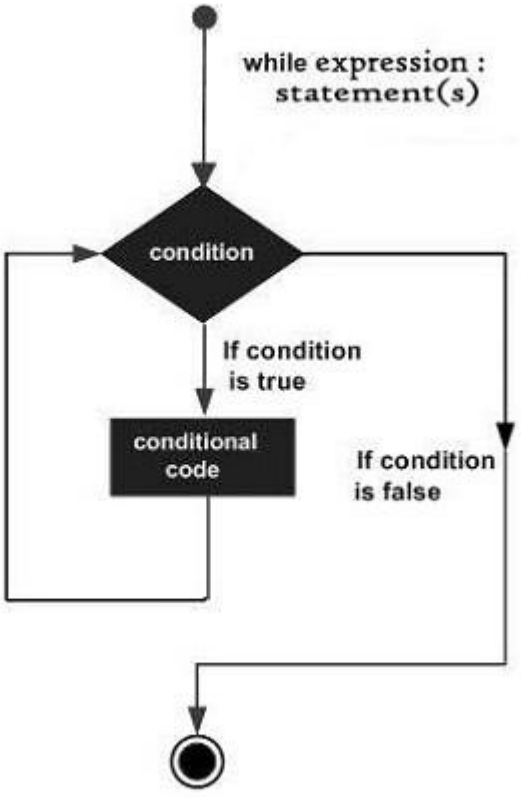
Here, statement(s) may be a single statement or a block of statements with uniform indent. The condition may be any expression, and true is any non-zero value. As soon as the expression becomes false, the program control passes to the line immediately following the loop.

If it fails to turn false, the loop continues to run, and doesn't stop unless forcefully stopped. Such a loop is called infinite loop, which is undesired in a computer program.
Syntax of while Loop

The syntax of a while loop in Python programming language is –

```
while expression:
  statement(s)
```

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.



Example:-

```
count=0
while count<5:
  count+=1
  print ("Iteration no. {}".format(count))
print ("End of while loop")
```

while-else Loop

Python supports having an **else statement** associated with a **while loop**. If the **else statement** is used with a **while loop**, the **else statement** is executed when the condition becomes false before the control shifts to the main line of execution.

Ex:-

```
count=0
while count<5:
  count+=1
  print ("Iteration no. {}".format(count))
else:
  print ("While loop over. Now in else block")
print ("End of while loop")
```

The **for loop**

The **for loop** in Python provides the ability to loop over the items of any sequence, such as a list, tuple or a string. It performs the same action on each item of the sequence. This loop starts with the **for** keyword, followed by a variable that represents the current item in the sequence. The **in** keyword links the variable to the sequence you want to iterate over. A **colon (:)** is used at the end of the loop header, and the indented block of code beneath it is executed once for each item in the sequence.

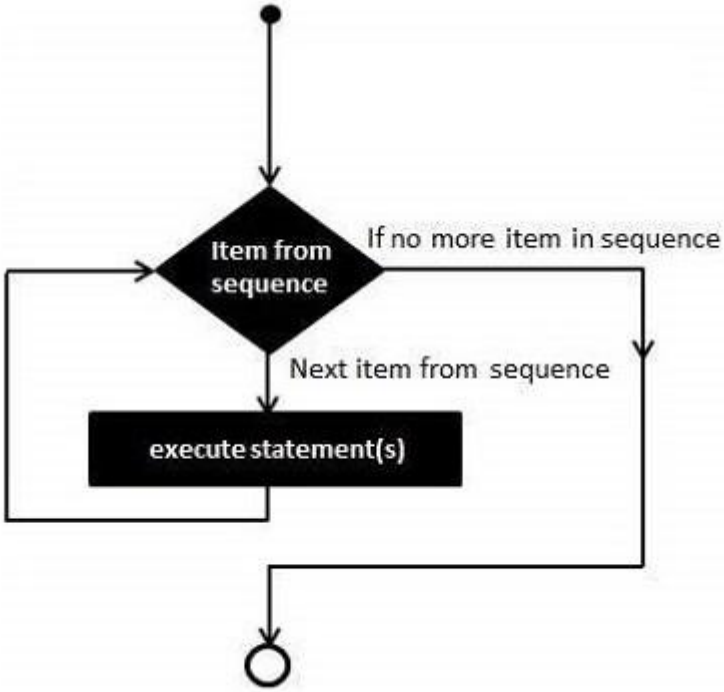
Syntax of Python for Loop

```
for iterating_var in sequence:
  statement(s)
```

Here, the **iterating_var** is a variable to which the value of each sequence item will be assigned during each iteration. **Statements** represents the block of code that you want to execute repeatedly.

Flowchart of Python for Loop

The following flow diagram illustrates the working of **for** loop –



for Loop with Strings

A string is a sequence of Unicode letters, each having a positional index. Since, it is a sequence, you can iterate over its characters using the for loop.

Example

The following example compares each character and displays if it is not a vowel ('a', 'e', 'i', 'o', 'u').

Example:

```
zen = ''' Beautiful is better than ugly.
 Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated. '''
for char in zen:
        if char not in 'aeiou':
                print (char, end='')
```

for Loop with Tuples

Python's tuple object is also an indexed sequence, and hence you can traverse its items with a for loop.

Example

In the following example, the for loop traverses a tuple containing integers and returns the total of all numbers.

Example:

```
numbers = (34,54,67,21,78,97,45,44,80,19)
total = 0
for num in numbers:
  total += num
print ("Total =", total)
```

for Loop with Range Objects

Python's built-in range() function returns an iterator object that streams a sequence of numbers. This object contains integers from start to stop, separated by step parameter. You can run a for loop with range as well.

Syntax

The range() function has the following syntax –

range(start, stop, step)

Where,

Start – Starting value of the range. Optional. Default is 0

Stop – The range goes upto stop-1

Step – Integers in the range increment by the step value. Option, default is 1.

Example

In this example, we will see the use of range with for loop.

```
for num in range(5):
  print (num, end=' ')
print()
for num in range(10, 20):
  print (num, end=' ')
print()
for num in range(1, 10, 2):
  print (num, end=' ')
```

Q4. Explain about break,continue,pass statements

Python break Statement

Python break statement is used to terminate the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for Python break statement is when some external condition is triggered requiring a sudden exit from a loop. The break statement can be used in both Python while and for loops.

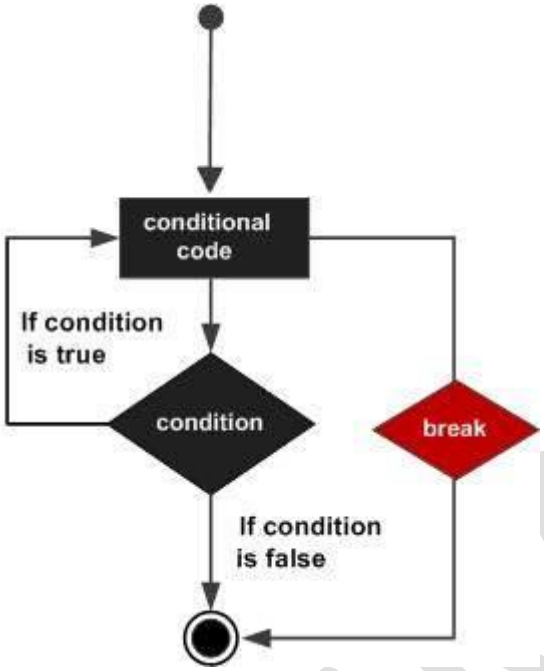
If you are using nested loops in Python, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

Syntax of break Statement

The syntax for a **break statement** in Python is as follows –

```
looping statement:
  condition check:
    break
```

Following is the flowchart of the break statement –



break Statement with for loop

If we use break statement inside a **for loop**, it interrupts the normal flow of program and exit the loop before completing the iteration.

Example

In this example, we will see the working of break statement in for loop.

```
for letter in 'Python':
  if letter == 'h':
    break
  print ("Current Letter :", letter)
print ("Good bye!")
```

Example

```
var = 10
while var > 0:
  print ('Current variable value :', var)
  var = var -1
  if var == 5:
    break

print ("Good bye!")
```

break Statement with Nested Loops

In nested loops, one loop is defined inside another. The loop that enclose another loop (i.e. inner loop) is called as outer loop.

When we use a break statement with nested loops, it behaves as follows –

- When break statement is used inside the inner loop, only the inner loop will be skipped and the program will continue executing statements after the inner loop
- And, when the break statement is used in the outer loop, both the outer and inner loops will be skipped and the program will continue executing statements immediate to the outer loop.

continue Statement

Python **continue statement** is used to skip the execution of the program block and returns the control to the beginning of the current loop to start the next iteration. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

The **continue statement** is just the opposite to that of break. It skips the remaining statements in the current loop and starts the next iteration.

of continue Statement

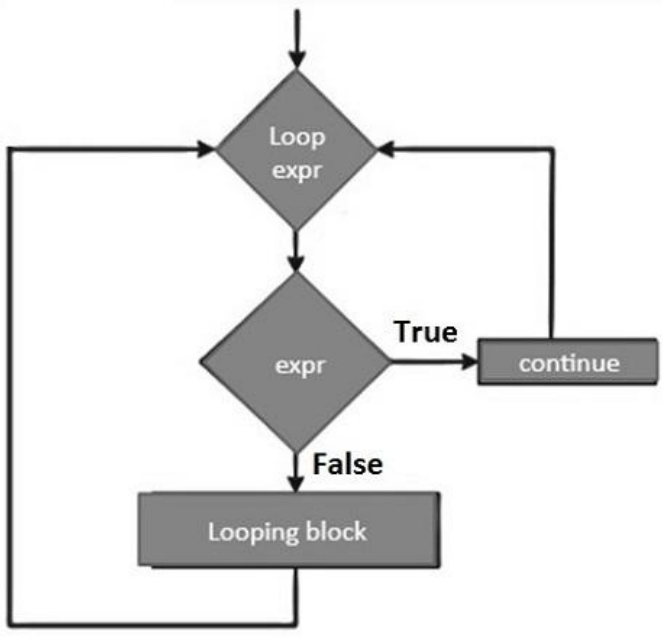
syntax:

looping statement:

condition check:

continue

Flow Diagram of continue Statement



continue Statement with for Loop

In Python, the continue statement is allowed to be used with a for loop. Inside the for loop, you should include an if statement to check for a specific condition. If the condition becomes TRUE, the continue statement will skip the current iteration and proceed with the next iteration of the loop.

Example

Let's see an example to understand how the continue statement works in for loop.

```
for letter in 'Python':
  if letter == 'h':
    continue
  print ('Current Letter :', letter)
print ("Good bye!")
```

pass Statement

Python pass statement is used when a statement is required syntactically but you do not want any command or code to execute. It is a null which means nothing happens when it executes. This is also useful in places where piece of code will be added later, but a placeholder is required to ensure the program runs without errors.

For instance, in a function or class definition where the implementation is yet to be written, pass statement can be used to avoid the SyntaxError. Additionally, it can also serve as a placeholder in control flow statements like for and while loops.

Syntax of pass Statement

Following is the syntax of Python **pass statement** −

```
Pass
```

Example of pass Statement

The following code shows how you can use the **pass statement** in Python −

```
for letter in 'Python':
  if letter == 'h':
    pass
    print ('This is pass block')
  print ('Current Letter :', letter)
print ("Good bye!")
```

Q5 Explain about usage of Arrays in Python?

Unlike other programming languages like C++ or Java, Python does not have built-in support for arrays. However, Python has several data types like lists and tuples (especially lists) that are often used as arrays but, items stored in these types of sequences need not be of the same type.

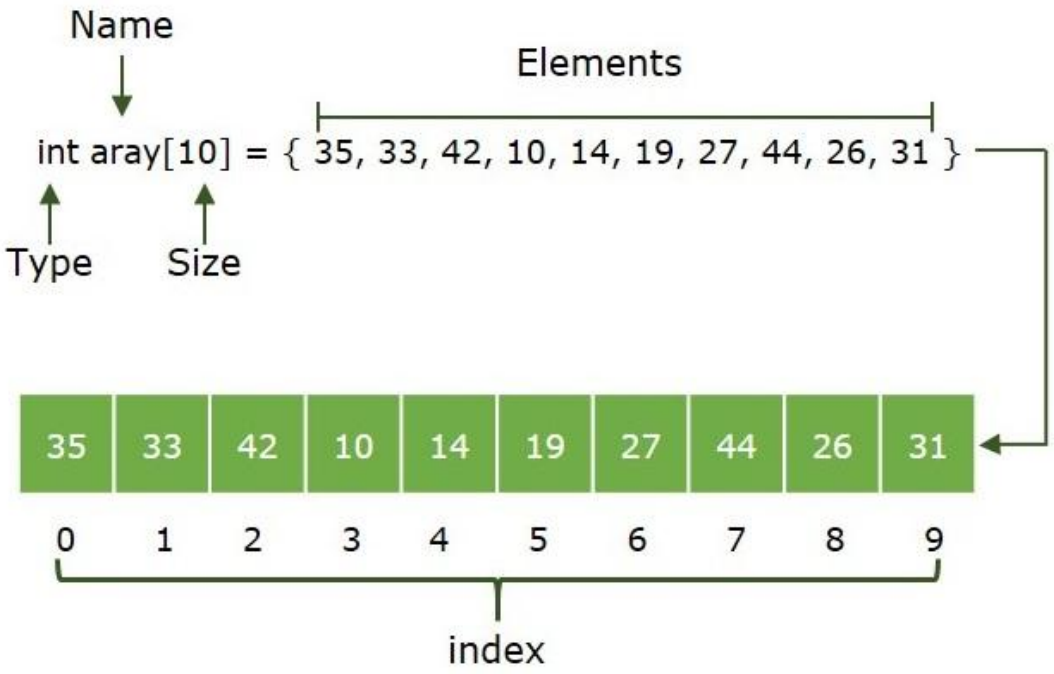
In addition, we can create and manipulate arrays the using **the array module.**

An array is a container which can hold a fix number of items and these items should be of the same type. Each item stored in an array is called an element and they can be of any type including integers, floats, strings, etc.

Array Representation

Arrays are represented as a collection of multiple containers where each container stores one element. These containers are indexed from '0' to 'n-1', where n is the size of that particular array.

Arrays can be declared in various ways in different languages. Below is an illustration –



following are the important points to be considered –

- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index.

a) Creating Array in Python

To create an array in Python, import the array module and use its **array() function**. We can create an array of three basic types namely integer, float and Unicode characters using this function.

The array() function accepts typecode and initializer as a parameter value and returns an object of array class.

Syntax

The syntax for creating an array in Python is –

```
# importing
import array as array_name

# creating array
obj = array_name.array(typecode[, initializer])
```

Where,

- **typecode** – The typecode character used to speccify the type of elements in the array.
- **initializer** – It is an optional value from which array is initialized. It must be a list, a bytes-like object, or iterable elements of the appropriate type.

```
import array as arr

# creating an array with integer type
a = arr.array('i', [1, 2, 3])
print (type(a), a)

# creating an array with char type
a = arr.array('u', 'BAT')
print (type(a), a)

# creating an array with float type
a = arr.array('d', [1.1, 2.2, 3.3])
print (type(a), a)
```

Python array type is decided by a single character Typecode argument. The type codes and the intended data type of array is listed below –

typecode	Python data type	Byte size
'b'	signed integer	1
'B'	unsigned integer	1
'u'	Unicode character	2

'h'	signed integer	2
'H'	unsigned integer	2
'i'	signed integer	2
'I'	unsigned integer	2
'l'	signed integer	4
'L'	unsigned integer	4
'q'	signed integer	8
'Q'	unsigned integer	8
'f'	floating point	4
'd'	floating point	8

Basic Operations on Python Arrays

Following are the basic operations supported by an array −

- **Traverse** − Print all the array elements one by one.
- **Insertion** − Adds an element at the given index.
- **Deletion** − Deletes an element at the given index.
- **Search** − Searches an element using the given index or by the value.
- **Update** − Updates an element at the given index.

Accessing Array Element

We can access each element of an array using the index of the element.

Example

The below code shows how to access elements of an array.

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1[0])
print (array1[2])
```

When we compile and execute the above program, it produces the following result −

```
10
30
```

Insertion Operation

In insertion operation, we insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.

Adding Elements to Python Array

There are multiple ways to add elements to an array in Python –

- Using append() method
- Using insert() method
- Using extend() method

Example

Here, we add a data element at the middle of the array using the python in-built insert() method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.insert(1,60)
for x in array1:
  print(x)
```

Ex:-

```
import array as arr
a = arr.array('i', [1, 2, 3])
a.append(10)
print (a)
```

Ex:-

```
import array as arr
 a = arr.array('i', [1, 2, 3, 4, 5])
 b = arr.array('i', [6,7,8,9,10])
 a.extend(b) print (a)
```

Deletion Operation

Deletion refers to removing an existing element from the array and re-organizing all elements of an array.

Here, we remove a data element at the middle of the array using the python in-built **remove()** method.

```
from array import *
array1 = array('i', [10,20,30,40,50])
array1.remove(40)
for x in array1:
```

```
print(x)
```

When we compile and execute the above program, it produces the following result which shows the element is removed form the array.

```
10
20
30
50
```

The array module defines two methods namely remove() and pop().

The remove() method removes the element by value whereas the pop() method removes array item by its position.

```
import array as arr
# creating array
numericArray = arr.array('i', [111, 211, 311, 411, 511])
# before removing array
print ("Before removing:", numericArray)
# removing array
numericArray.remove(311)
# after removing array
print ("After removing:", numericArray)
```

Search Operation

You can perform a search operation on an array to find an array element based on its value or its index.

Example

Here, we search a data element using the python in-built index() method −

```
from array import *
array1 = array('i', [10,20,30,40,50])
print (array1.index(40))
```

When we compile and execute the above program, it will display the index of the searched element. If the value is not present in the array, it will return an error.

```
3
```

Update Operation

Update operation refers to updating an existing element from the array at a given index. Here, we simply reassign a new value to the desired index we want to update.

Example

In this example, we are updating the value of array element at index 2.

```
from array import *
```

```
array1 = array('i', [10,20,30,40,50])
array1[2] = 80
for x in array1:
  print(x)
```

On executing the above program, it produces the following result which shows the new value at the index position 2.

```
10
20
80
40
50
```

Reverse an Array in Python

To reverse an array, use the following approaches –

- Using slicing operation
- Using reverse() method
- Using reversed() method
- Using for loop

Using slicing operation

Slicing operation is the process of extracting a part of array within the specified indices. In Python, if we use the slice operation in the form **[::-1]** then, it will display a new array by reversing the original one.

```
import array as arr

# creating array
numericArray = arr.array('i', [88, 99, 77, 55, 66])

print("Original array:", numericArray)
revArray = numericArray[::-1]
print("Reversed array:",revArray)
```

Reverse an Array Using reverse() Method

We can also reverse the sequence of numbers in an array using the reverse() method of list class. Here, list is a built-in type in Python.

Since reverse() is a method of list class, we cannot directly use it to reverse an array created through the Python array module. We have to first transfer the contents of an

Array to a list with **tolist()** method of array class, then we call the reverse() method and at the end, when we convert the list back to an array, we get the array with reversed order.

```
import array as arr
# creating an array
numericArray = arr.array('i', [10,5,15,4,6,20,9])
print("Array before reversing:", numericArray)
# converting the array into list
newArray = numericArray.tolist()
# reversing the list
newArray.reverse()
```

Sorting Arrays

Python's array module defines the array class. An object of array class is similar to the array as present in Java or C/C++. Unlike the built-in Python sequences, array is a homogenous collection of either strings, or integers, or float objects.

The array class doesn't have any function/method to give a sorted arrangement of its elements. However, we can achieve it with one of the following approaches –

- Using a sorting algorithm
- Using the sort() method from List
- Using the built-in sorted() function

Sort Arrays Using sort() Method of List

Even though array module doesn't have a sort() method, Python's built-in List class does have a sort method. We shall use it in the next example.

First, declare an array and obtain a list object from it, using tolist() method. Then, use the sort() method to get a sorted list. Lastly, create another array using the sorted list which will display a sorted array.

Example

The following code shows how to get sorted array using the sort() method.

```
import array as arr
# creating array
orgnlArray = arr.array('i', [10,5,15,4,6,20,9])
print("Original array:", orgnlArray)
# converting to list
```

```
sortedList = orgnlArray.tolist()
# sorting the list
sortedList.sort()

# creating array from sorted list
sortedArray = arr.array('i', sortedList)
print("Array after sorting:",sortedArray)
```

The above code will display the following output −

```
Original array: array('i', [10, 5, 15, 4, 6, 20, 9])
Array after sorting: array('i', [4, 5, 6, 9, 10, 15, 20])
```

Sort Arrays Using sorted() Method

The third technique to sort an array is with the sorted() function, which is a built-in function.

The syntax of sorted() function is as follows −

```
sorted(iterable, reverse=False)
```

The function returns a new list containing all items from the iterable in ascending order.
Set reverse parameter to True to get a descending order of items.
The sorted() function can be used along with any iterable.
Python array is an iterable as it is an indexed collection. Hence, an array can be used as a parameter to sorted() function.

Ex:-

```
import array as arr
a = arr.array('i', [4, 5, 6, 9, 10, 15, 20])
sorted(a)
print(a)
```

Join two Arrays in Python

To join arrays in Python, use the following approaches −

- Using append() method
- Using + operator

Using append() Method

To join two arrays, we can append each item from one array to other using append() method. To perform this operation, run a for loop on the original array, fetch each element and append it to a new array.

Example: Join Two Arrays by Appending Elements

Here, we use the append() method to join two arrays.

```
import array as arr

# creating two arrays
a = arr.array('i', [10,5,15,4,6,20,9])
b = arr.array('i', [2,7,8,11,3,10])

# merging both arrays
for i in range(len(b)):
  a.append(b[i])
print (a)
```

Using + operator

We can also use **+ operator** to concatenate or merge two arrays. In this approach, we first convert arrays to list objects, then concatenate the lists using the + operator and convert back to get merged array.

Example: Join Two Arrays by Converting to List

```
import array as arr
a = arr.array('i', [10,5,15,4,6,20,9])
b = arr.array('i', [2,7,8,11,3,10])
x = a.tolist()
y = b.tolist()
z = x+y
a = arr.array('i', z)
print (a)
```