BCA SEM-VI

Subject: Python Programming (BCA-601)

Created by: Smit Trivedi (Asst.Prof BCA/PGDCA College Idar)

UNIT 2

Topic: Control statements

Looping statements

Arrays

Functions

Modules

Control statements

- Control statements in Python **are used to direct the flow of a Program’s execution.**

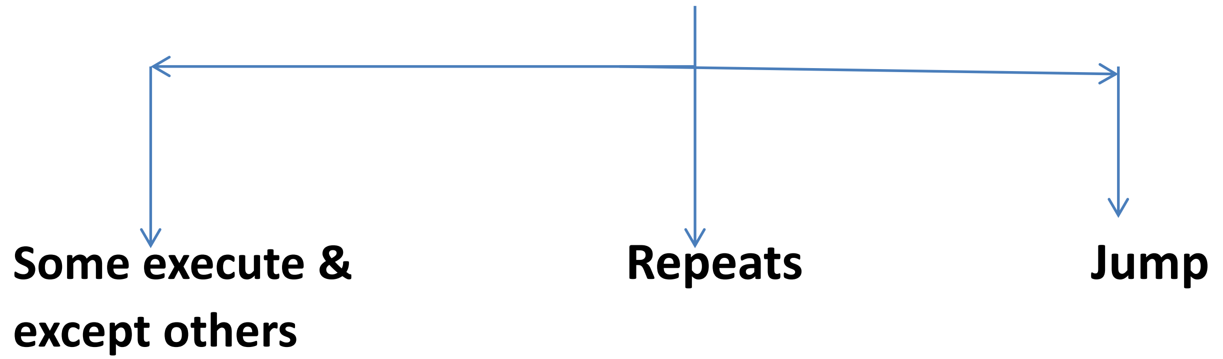
  means program (top to bottom)

  print(“Pass”)

  print(“Fail”)

  Mainly Three concept to all programming language supported.

Control statements



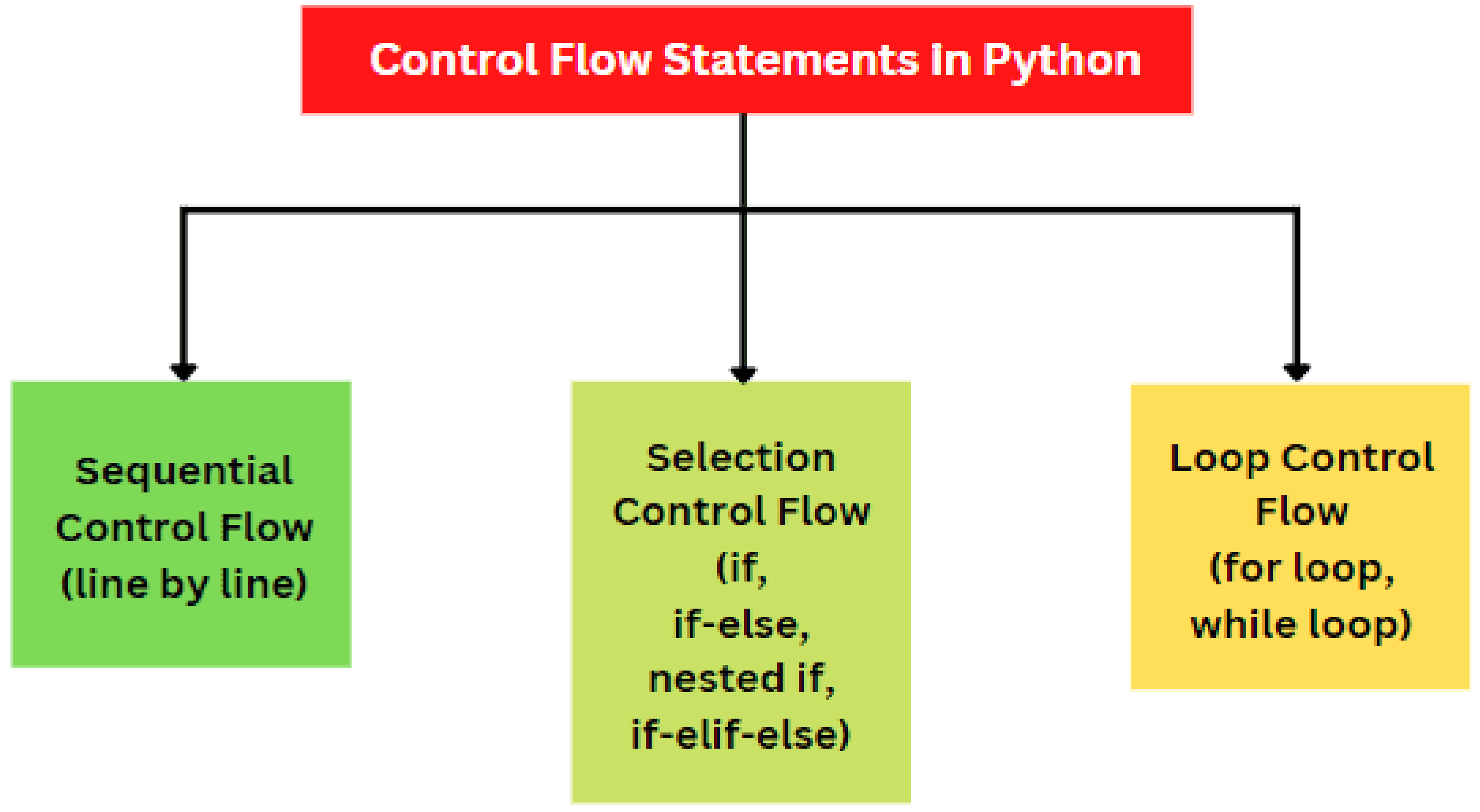

Control Flow Statements in Python
Sequential Control Flow (line by line)
Selection Control Flow (if, if-else, nested if, if-elif-else)
Loop Control Flow (for loop, while loop)

Fig: Types of control flow statements in Python

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

Equals: a == b

Not Equals: a != b

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

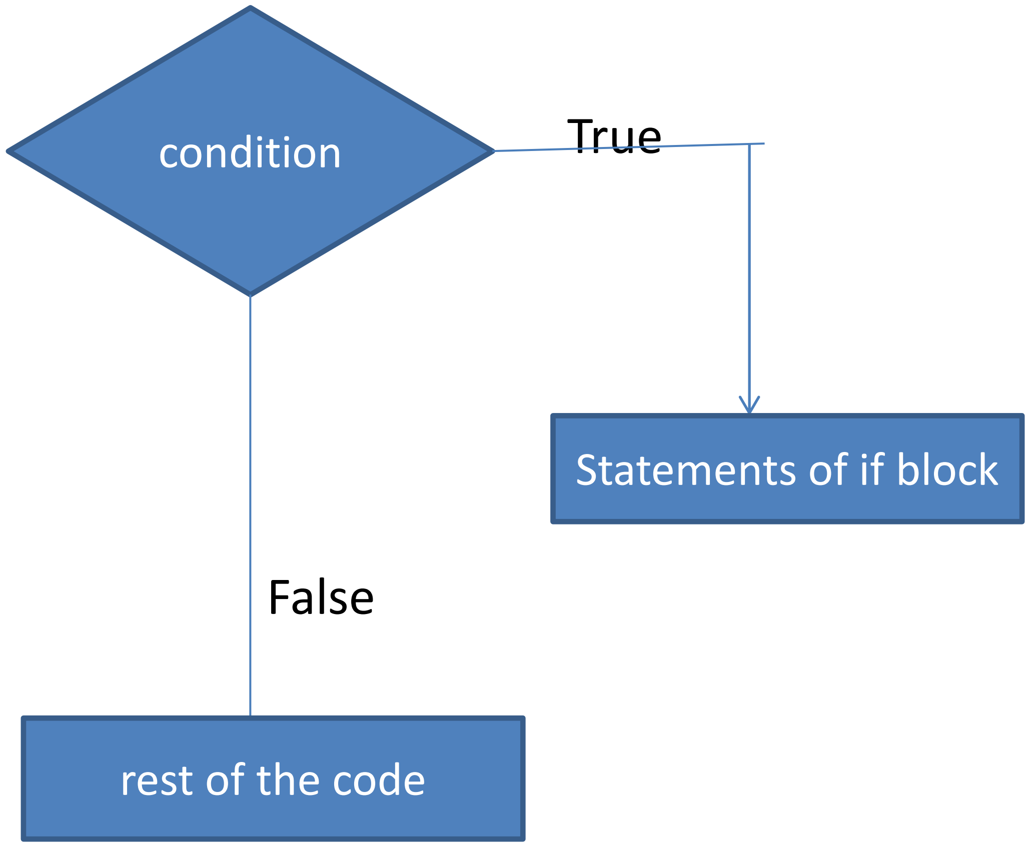
These conditions can be used in several ways, most commonly in "if statements" and loops.

Example:

```
a = 30
b = 20
if a > b:
print(“a is greter than b”)
```

In this example we use two variables, a and b, which are used as part of the if statement to test whether a is greater than b. As a is 30, and b is 20, we know that 30 is greater than 20, and so we print to screen that "a is greater than b".

If Statement: is used for decision-making operations. It contains a body of code which runs only when the condition given in the if statement is true. If the condition is false, then the optional else statement runs which contains some code for the else condition.



Syntax

if(expression==true)

Statement(body of statement)

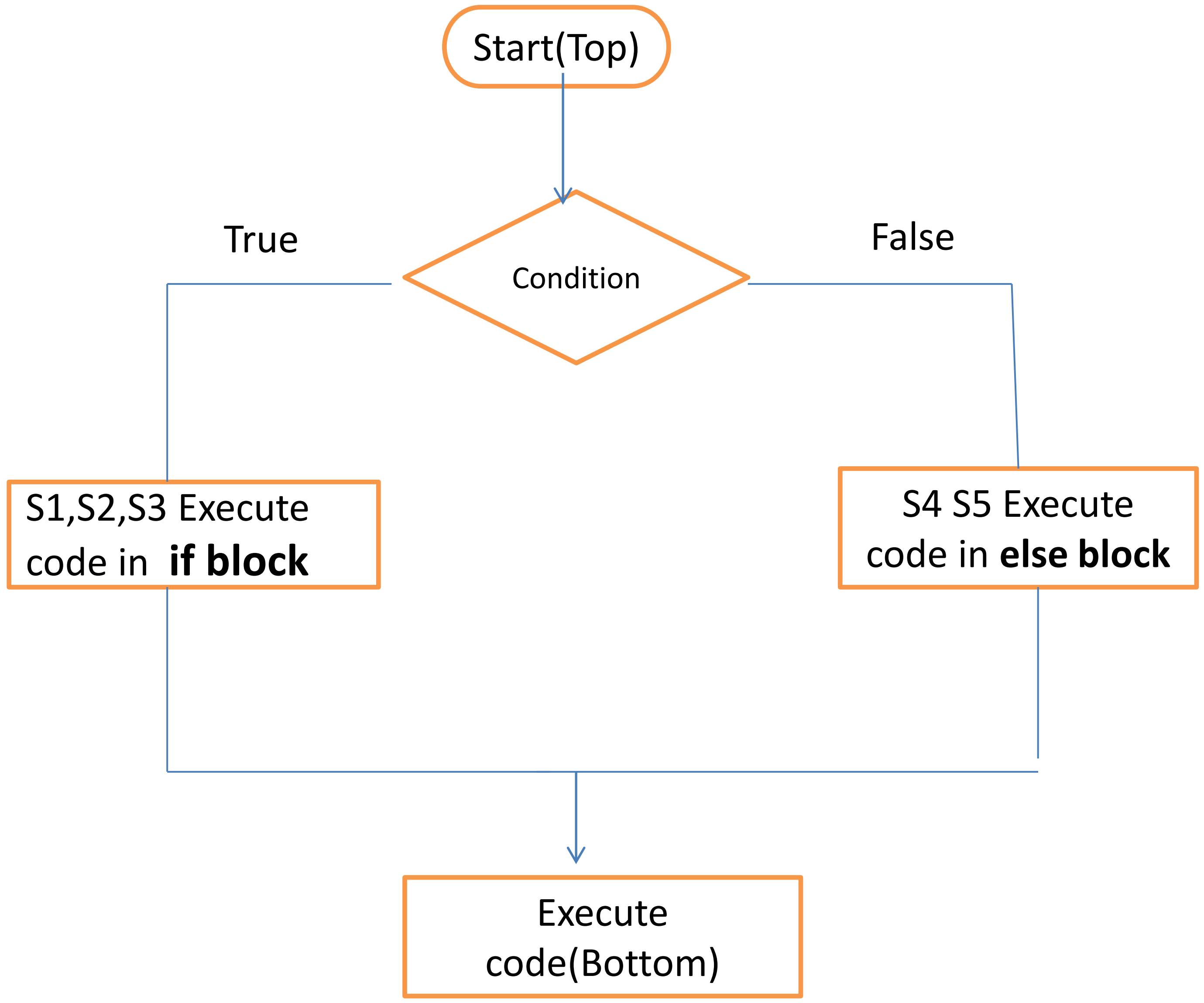
If Else Statement

"If " & "Else" are Keyword.

Bibranching Two Way of concept.

if-else statement in Python allows you to execute a block of code based on a condition. It is a decision-making statement that executes one block of code if a condition is true and another block if the condition is false.

```
Syntax: If (Condition): (Expression == True):
             s1,s2,s3 (statement of block)
            Else:
             s4,s5  (statement of block)
```


Start(Top)
Condition
True
False
S1,S2,S3 Execute code in **if block**
S4 S5 Execute code in **else block**
Execute code(Bottom)

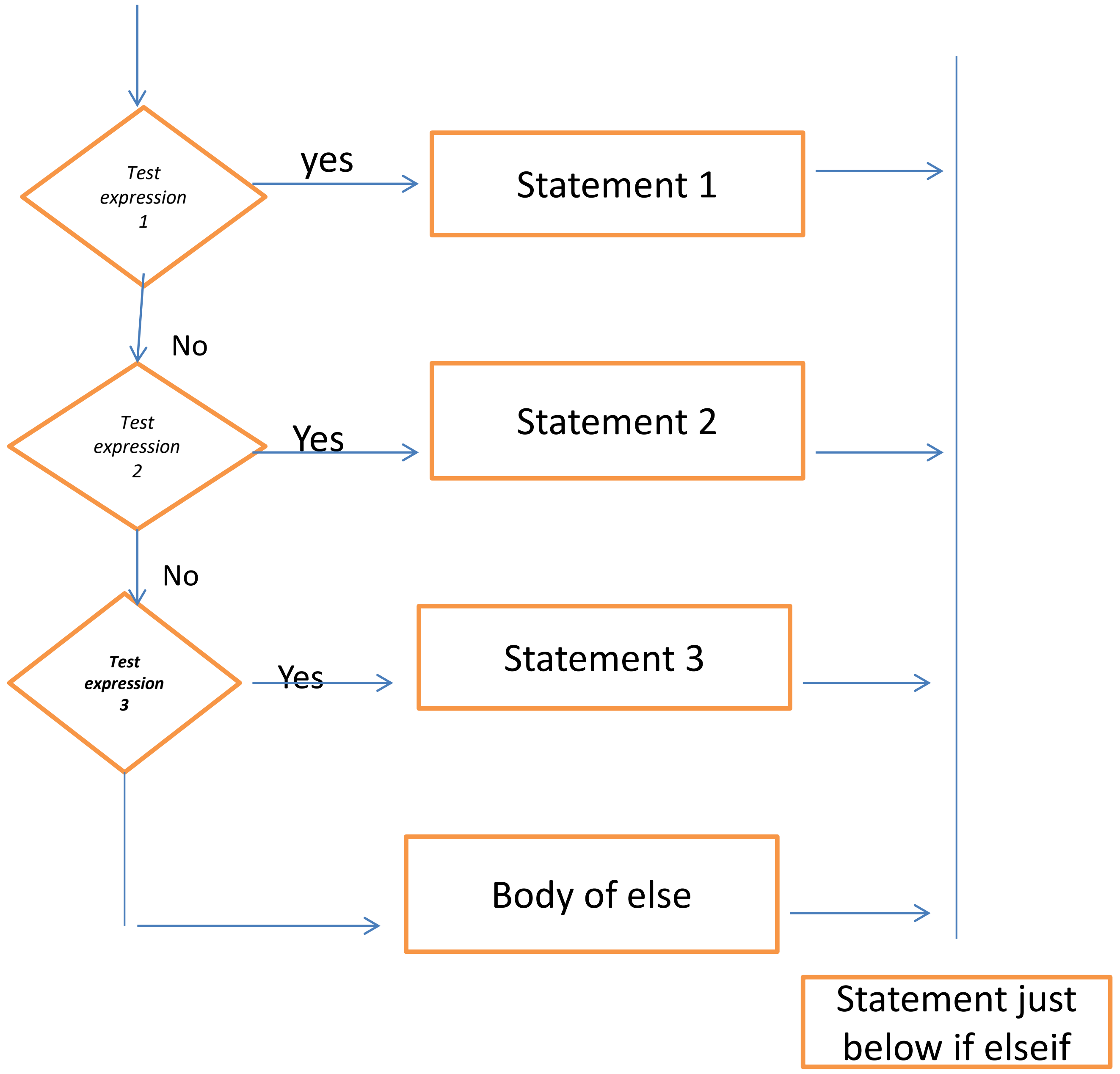
- Example:
- Mark = int (input("Enter Marks:"))
- if(Mark>40):
-     print("Pass")
- else:
-     print("Fail")
-     print("Thanks")

If- elif statement

To show a Multi way decision based on several condition we use **if elif else** statement.

The if-elif statement is shortcut of if..else chain. While using if-elif statement at the end else block is added which is performed if none of the above **if-elif** statement is true.

```
Syntax: if (condition):
                statement elif (condition):
                   statement…..
                       else:
                     statement
```


Test expression 1
yes
Statement 1
No
Test expression 2
Yes
Statement 2
No
Test expression 3
Yes
Statement 3
Body of else
Statement just below if elseif

```
If (Condition 1):
      Statement 1
Elif (Condition 2):
      Statement 2
Elif (Condition 3):
     Statement 3
Elif (Condition n):
     Statement n
Else:
    Statement x
  Rest of the Code
```

Looping

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.

Programming languages provide various control structures that allow for more complicated execution paths.

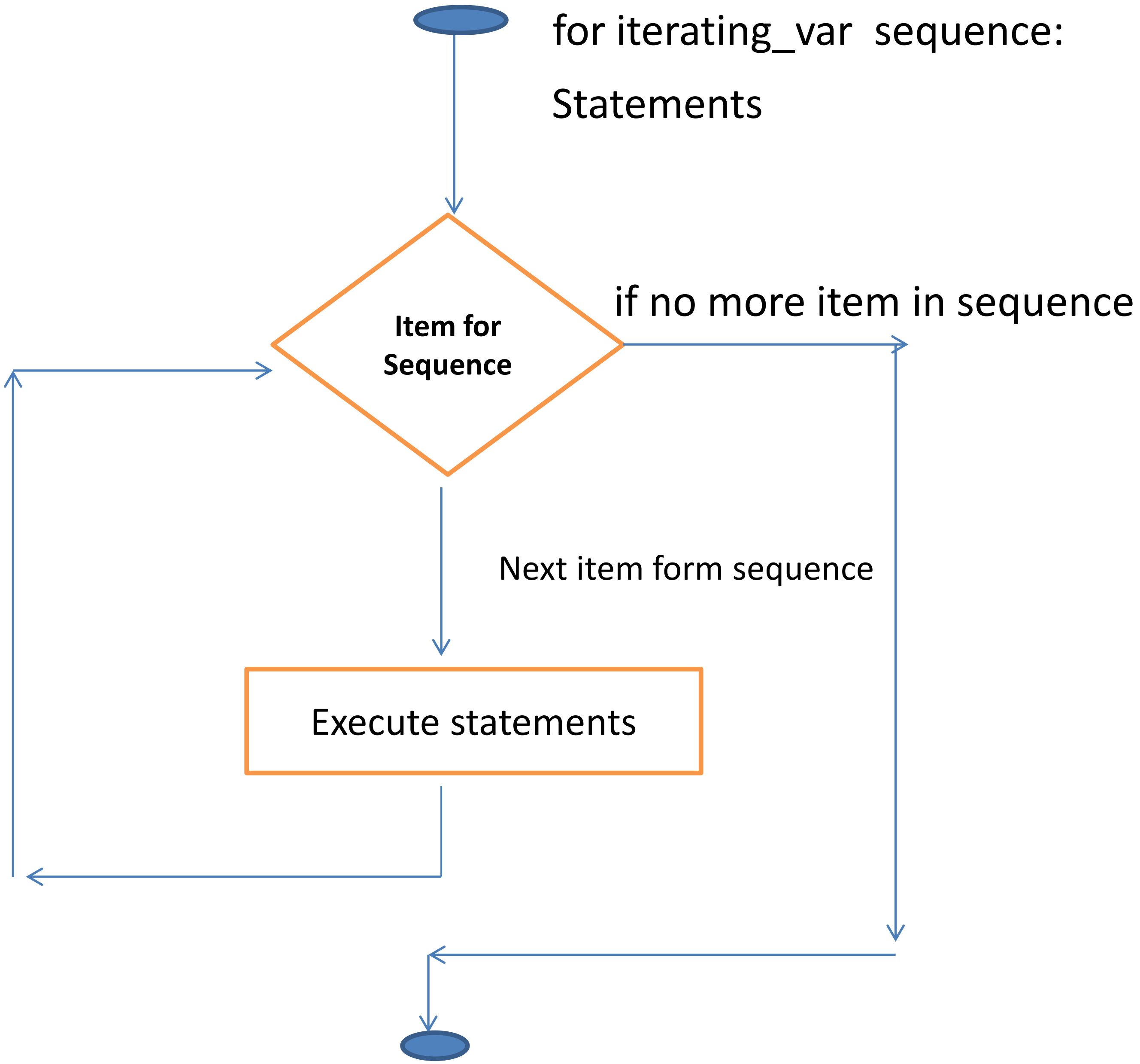
A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement −

Python has two primitive loop commands:

(1) For Loops (definite) (2) While Loops (indefinite)

for iterating_var sequence:

Statements



Syntax : For i in range(start,stop,step):

statements

Example :

```
for i in range(1,10)   # 1 is include or 10 is exclude.
print(i)
```

Example : **Loop Through a String**

```
language = 'Python'
# iterate over each character in language
for x in language:
print(x)
```

While loop: A loop that has indefinite number of times to run.

In other words, a while loop repeats the body of the loop as long as a given condition is true.

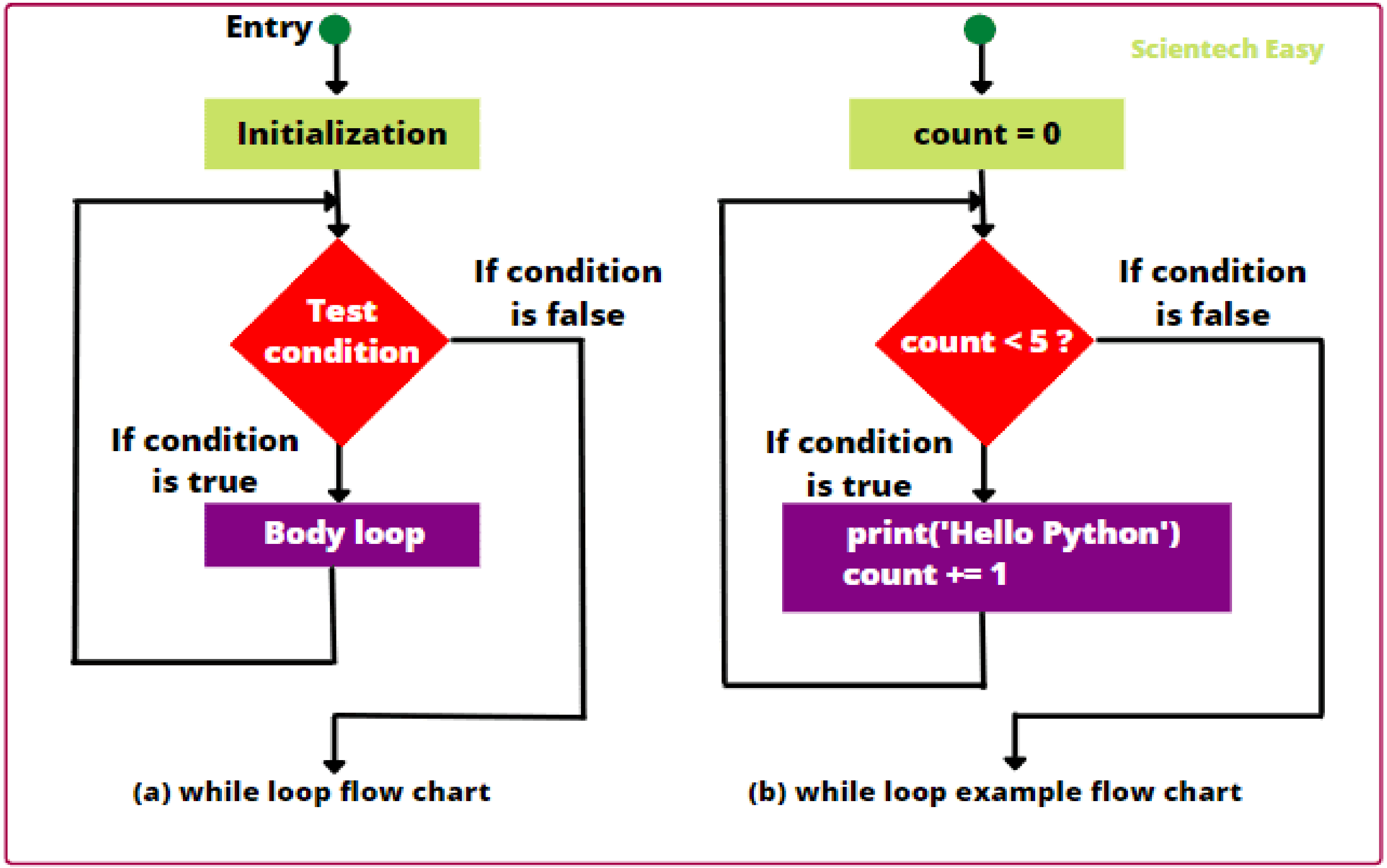
It evaluates the condition each time it executes the loop body, and it exits the loop when the condition becomes false. Each repetition of the body of loop is called **iteration of loop**.

If the condition is false, then the loop body does not execute. In this way, the while loop statement executes a block of code repeatedly while the test condition is true.

Syntax: while Test Condition:

```
            #Loop of body
            statement(s)
```

While loop Flow chart



(a) while loop flow chart

(b) while loop example flow chart

Way to Three Statements use to loops

1. **Break Statements**
2. **Continue Statements**
3. **else**

Example: 1 # Program to display numbers from 1 to 5.

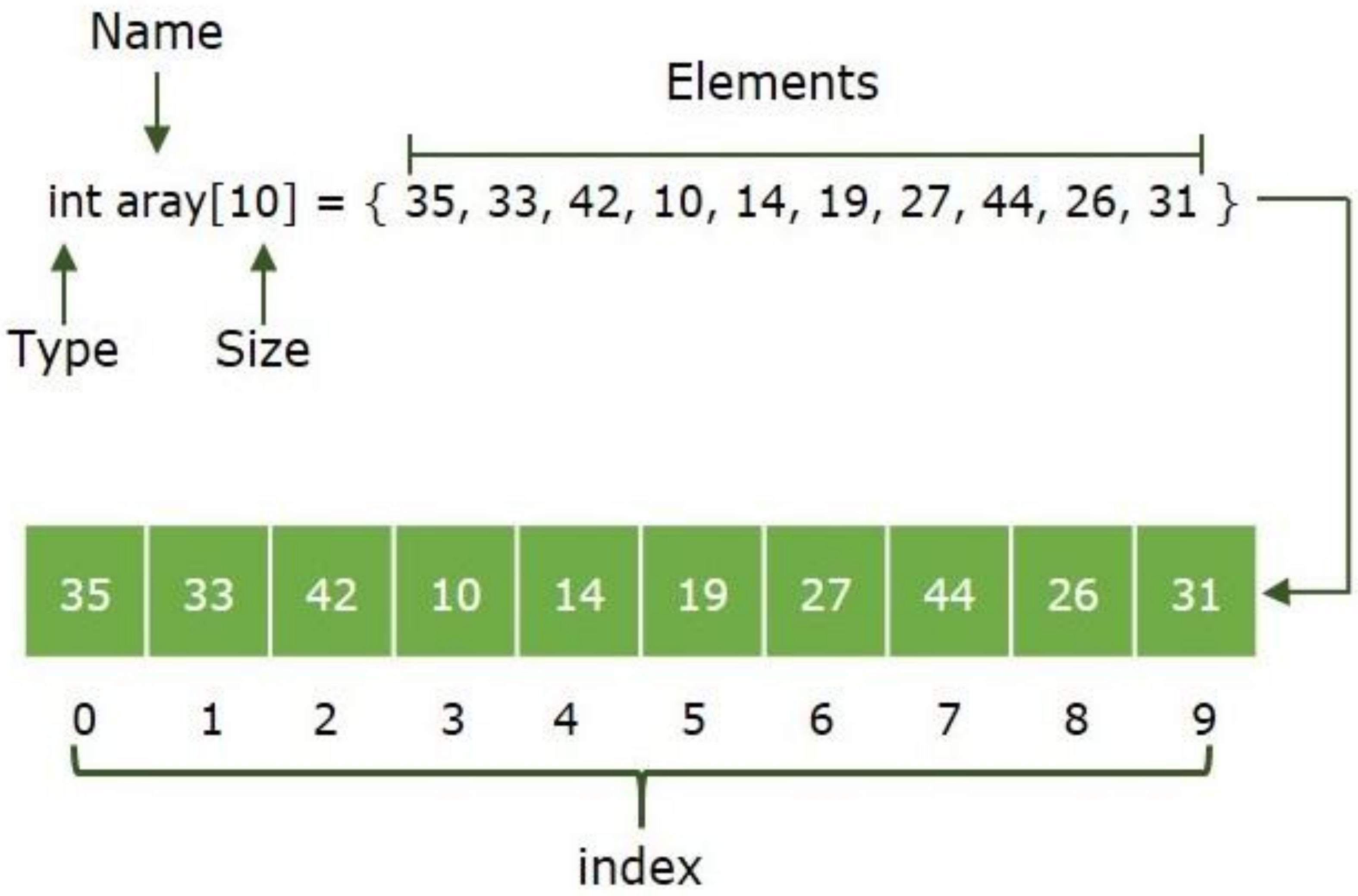
```
i= 1 # Initialization.
while i<= 5:
print('Current value of i is ',i)
i = i + 1 # Increment by 1.
```

Example: 2 *# Program to display your name five times.*

```
i = 0
Name = input(‘Enter your Name:’)
while I < 5:
print (Name)
i= i + 1
```

Arrays

- An array is defined as a collection of items that are stored at contiguous memory locations. It is a container which can hold a fixed number of items, and these items should be of the same type. An array is popular in most programming languages like C/C++, JavaScript, Python 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.
- Index starts with 0.
- Array length is 10 which means it can store 10 elements.
- Each element can be accessed via its index. For example, we can fetch an element at index 6 as 9.



Name
Elements
int aray[10] = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 }
Type
Size
35
33
42
10
14
19
27
44
26
31
0
1
2
3
4
5
6
7
8
9
index

Advantages of the arrays

1. Efficient access and handling.
2. Versatility with different data types
3. Built in functions for manipulation
4. Optimized memory usage
5. Fast data retrieval
6. Supports various data structures
7. Simplified iteration
8. Widely used and supported..

One Dimensional Array: (1D Array)

One Dimensional Array using Array module..

Single Row or Multiple Columns..

For Ex:[101,102,103,104,105]

★ **Import Array Module**

Two way to import array module:

(1: import array- this will import the entire array module.

(2: from array import * this will import all class,objects,variable etc from array module. here * means all..

Creating an array

- 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 type code and initializer as a parameter value and returns an object of array class.

Syntax: # importing

```
import array as array _name
# creating array
obj = array_name.array(typecode[, intializer])
```

typecode – The typecode character used to specify 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.

- Example:

```
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)
```

Accessing Array Element

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

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

Looping array

- Loops are used to repeatedly execute a block of code.
- There are two types of loops named: **for loop and while loop.**
- Since the array object behaves like a sequence, you can iterate through its elements with the help of loops.
- arrays is to perform operations such as accessing, modifying, searching, or aggregating elements of the array.

(1) **for loop** is used when the number of iterations is known. If we use it with an iterable like array, the iteration continues until it has iterated over every element in the array.

Example: **import array as arr**

```
newArray = arr.array('i', [56, 42, 23, 85, 45])
for iterate in newArray:
print (iterate)
```

(2) while Loop with Array:

In **while loop**, the iteration continues as long as the specified condition is true.

When you are using this loop with arrays, initialize a loop variable before entering the loop.

This variable often represents an index for accessing elements in the array. Inside the while loop, iterate over the array elements and manually update the loop variable.

- Following example while loop:

```
import array as arr
# creating array
a = arr.array('i', [96, 26, 56, 76, 46])
# checking the length
i = len(a)
# loop variable
idx = 0
# while loop
while idx < i:
 print (a[idx])
# incrementing the while loop
 idx+=1
```

- **for Loop with Array Index**

We can find the length of array with built-in len() function.

Use it to create a range object to get the series of indices and then access the array elements in a **for** loop.

Example:

```
import array as arr
a = arr.array('d', [56, 42, 23, 85, 45])
l = len(a)
for x in range(l):
print (a[x])
```

Slicing and Processing the arrays

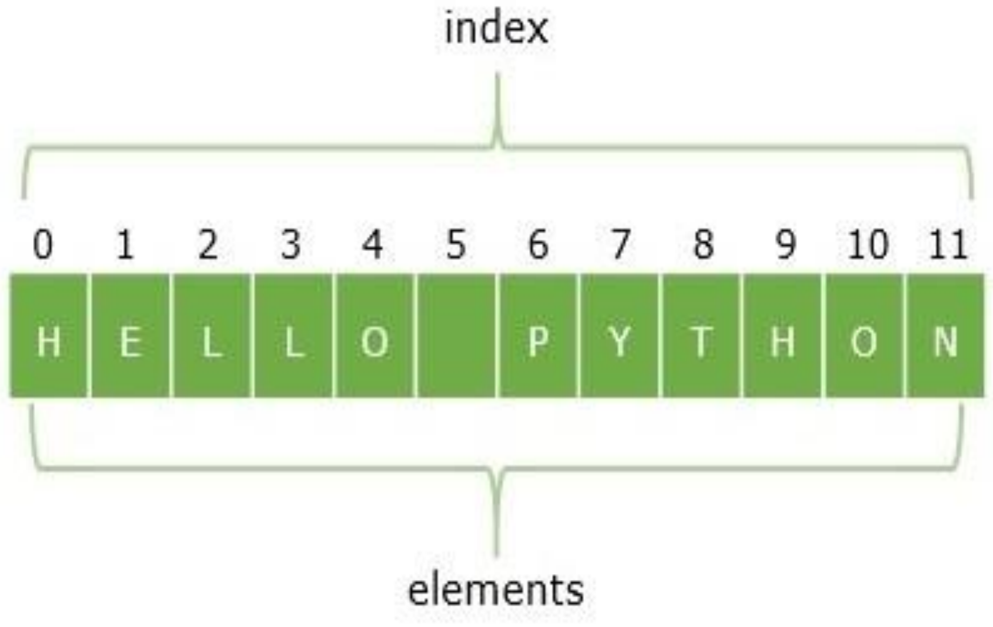
- Python String slicing is a way of creating a sub-string from a given string. In this process, we extract a portion or piece of a string. Usually, we use the slice operator "[:]" to perform slicing on a Python String.

  If a string variable is declared as

  var="HELLO PYTHON",

  Python allows you to access any individual character from the string by its index.

  In this case, 0 is the lower bound and 11 is the upper bound of the string.



- var = "HELLO PYTHON"
- print(var[0])
- print(var[7])
- print(var[11])
- print(var[12])

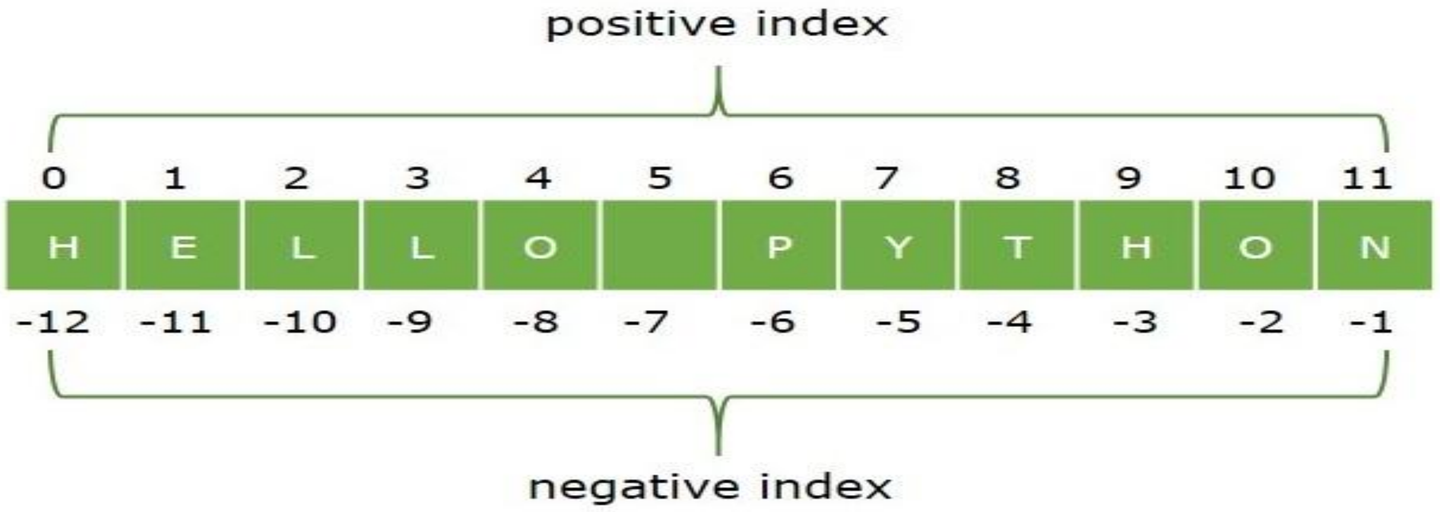
- Slicing in python String: [Start,Stop,Step]
- String1 = “Smit Trivedi”

  subset = string1[5:10]

★ Python String Negative & Positive Indexing

One of the unique features of Python sequence types. is that it has a negative indexing scheme also.

Example: a positive indexing scheme is used where the index increments from left to right. In case of negative indexing, the character at the end has -1 index and the index decrements from right to left, as a result the first character H has -12 index.



- var = "HELLO PYTHON“
- print(var[-1])
- print(var[-5])
- print(var[-12])

[Start,Stop,Step]

String1 = “Hello Python”

Substr2= string1[:10] **Omit value.. (by default Zero start)**

Substr3= string1[6:]   **(Value Start from Six : End of value**)

Substr4= string1[:]   (**All Value Defined)**

Substr5= string1[6::2] **Jump the value at a position(one by Step value)**

Substr6= string1[::-1] (**Reverse the value)**

Substr7= string1[6:11].upper()

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.

  Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Defining a Function:

(1) You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

(2) Function blocks begin with the keyword def followed by the function name and parentheses (( )).

(3) The code block within every function starts with a colon (:) and is indented.

- Syntax : def functionname(parameters): "function_docstring"

  function_suite

  return [expression]

Create a Function: In Python a function is defined using the def keyword:

```
def my_function():
 print("Hello How are you")
```

Calling and Returning(single and multiple) results from a function.

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- you can call it by using the function name itself.
- If the function requires any parameters, they should be passed within parentheses. If the function doesn't require any parameters, the parentheses should be left empty.
- **Example to Call a Python Function**

```
def fun ():
   return 1,2,3,4,5
    x = fun
    x
```

- Output: (1,2,3,4,5)

Return Statement(Single and Multiple)

Return statement can be used to return something from the function. in python, it is possible to return one or more variables/values.

Syntax (variable or expression);

single Value return

```
def add (y):
x = 10
return x + y
sum = add (20)
print (sum)
```

# **add** function returns the sum of x and y as a single value.

- Multiple value Return

```
 def add_ sub (y):
 x = 10
c = x+y
d = y-x
return c,d
sum, sub = add(20)
print(sum)
print(sub)
```

Task 1

```
def add(a, b):
return a + b
result = add(5, 7)
print(result)
```

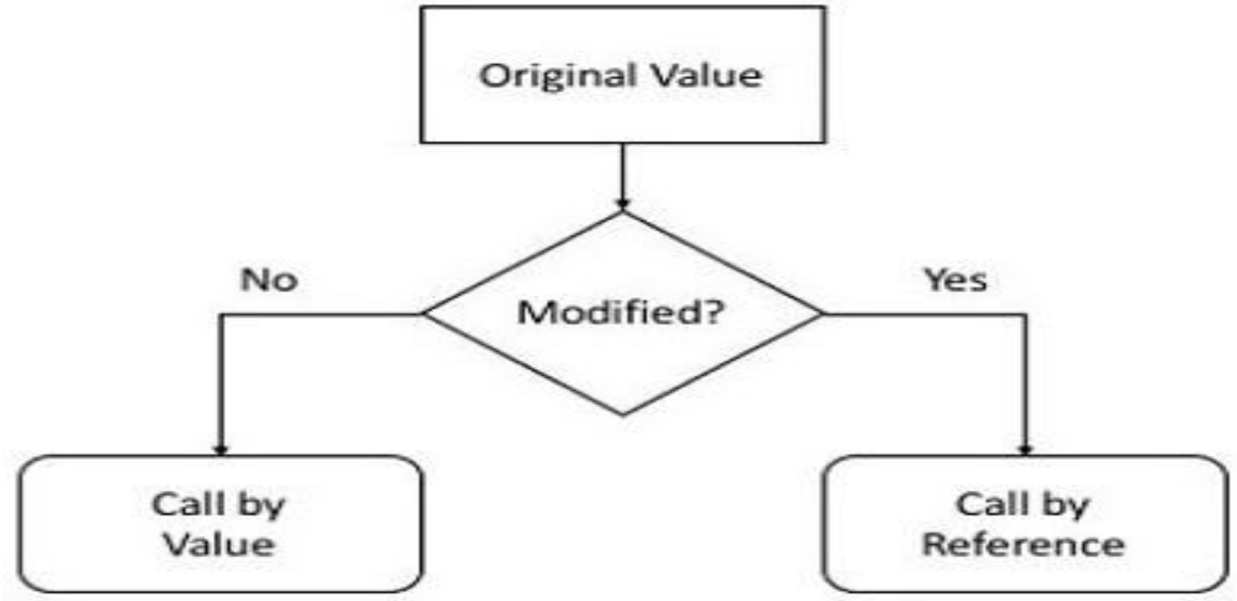
Task 2

```
def get_user_info():
 name = “ Prem"
age = 28
city = “Mumbai"
return name, age, city
info = get_user_info()
 print(info)
```

Pass by Object Reference

- In python, Neither of these two concepts is applicable rather the values are sent to function by means object reference.
- When we pass value like number,strings,tuple or lists to function, the references of these objects are passed to function.
- For this example:

```
x = 10    #  created to Object
val(x)
print(x, id(x))
```



call by reference – In this way of passing variable, a reference to the object in memory is passed. Both the formal arguments and the actual arguments (variables in the calling code) refer to the same object. Hence, any changes in formal arguments does get reflected in the actual argument.

Positional arguments

- **Positional arguments** are arguments passed to a function based on their position in the function call. The order in which these arguments are passed is crucial because Python assigns values to the parameters in the same order they appear in the function definition.
- Some of Example:

```
#positions must be maintained
 def power(a,b):
   print(a ** b)
# calling the Function with Positional arguments.
  power(2,3) #8
# Position changes
 power(3,2) #9
```

Keyword arguments

- **Keyword arguments** are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.
- Some of Example:

```
def printinfo( name, age ):
print ("Name: ", name)
print ("Age ", age)
return;
 # Now you can call printinfo function
  printinfo( age=28, name=“Smit" )
```

Default arguments

A **default argument** is an argument that assumes a default value if a value is not provided in the function call for that argument.

Syntax:

```
def example(arg1, arg2="default"):
Pass
```

Example:

```
def add(a,b,c=5):
d=a+b+c
print(“addition=“, d )
add(5,6,7)  #Function call
```

Variable Length arguments

Keyword Variable Length argument is an argument

That can accept any number of values provided in

the form of key value pair.

The keyword variable length argument is written with *(asterisk) symbol

It stores all the value in a dictionary in the form of key-value pair.

- Variable-length arguments, varargs for short, are arguments that can take an unspecified amount of input. When these are used, the programmer does not need to wrap the data in a list or an alternative sequence.
- In Python, varargs are defined using the *args
- syntax. Let's reimplement our my_min() function with *args:

Python Lambda (Anonymous Function)

A Function without a name is called as Anonymous Function. It is also known as Lambda Function.

→ Anonymous Function are not defined using def keyword rather they are defined using lambda keyword.

Syntax: **lambda arguments: expression**

or

lambda argument_list : expression

Ex: lambda x : print(x)
lambda x,y : x+y

❖ **Calling Lambda Function**

```
Sum = lambda x : x+1
Sum(5)

add = lambda x,y : x+y
add(5,2)
```

Why do we need a Lambda Function?

- When compared to a normal Python function written using the **def** keyword, lambda functions require fewer lines of code. However, this is not quite true because functions defined using def can be defined in a single line. But, **def** functions are usually defined on more than one line.
- They are typically employed when a function is required for a shorter period (temporary), often to be utilized inside another function such as filter, map, or reduce.
- You can define a function and call it immediately at the end of the definition using the lambda function. This is not possible with def functions.

Modules

What is Module?

Modules are Nothing but group of function, variables and class that are saved to a file.

Note: i) Code reusability.

ii) if python file (first.py) called that is module then its module name should be first.

There are two Types of Modules

(1) Pre-Define Module

(2) User Define Module

Pre- Define Module:	User Define Module
Calendar	First
Random	Ankush
Keyword	Ankit

Example of User Define:

First.py

```
def show():
print(‘hello’)
```

second.py

```
import first
first.show()
```