Unit – III

Function:-

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the python program.

In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions which can be called user-defined functions.

Need of function:-

- ***By using functions, we can avoid rewriting same logic/code again and again in a program.***
- ***We can call python functions any number of times in a program and from any place in a program.***
- ***We can track a large python program easily when it is divided into multiple functions.***
- ***Reusability is the main achievement of python functions.***
- ***However, Function calling is always overhead in a python program.***

Defining a Function

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

- ***Function blocks begin with the keyword def followed by the function name and parentheses (( )).***
- ***Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.***
- ***The first statement of a function can be an optional statement - the documentation string of the function or doc string.***
- ***The code block within every function starts with a colon (:) and is indented.***
- ***The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.***

Syntax:-

```
def  function-name(parameter):
        Instruction to be processed.
        Return statement
```

Examples

Ex 1

```
def hello_world():
        print("hello world")

# Now you can call printme function
hello_world()
```

Ex 2

```
#defining the function
def func (name):
        print("Hi ",name);

#calling the function
func("Ayush")
```

Ex 3

```
def sum (a,b):
        return a+b;

#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))

#printing the sum of a and b
print("Sum = ",sum(a,b))
```

Ex 4

```
def printme( str ):
        print str
        return;

# Now you can call printme function
printme("first call to user defined function!")
printme(" second call to the same function")
```


- *Parameter part in function is optional it may or may not be used.*
- *All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function*
- *There is an exception in the case of mutable objects since the changes made to the mutable objects like string do not revert to the original string rather, a new string object is made, and therefore the two different objects are printed.*

Ex 1

```
#defining the function immutable items def
change_list(list1):
   list1.append(20);
   list1.append(30);
   print("list inside function = ",list1)

#defining the list list1
= [10,30,40,50]

#calling the function
change_list(list1);
print("list outside function = ",list1);
```

Ex 2

```
#defining the function mutable objects like
(string)
def change_string (str): str
   = str + " Hows you";
   print("printing the string inside function
```

```
    :",str);

    string1 = "Hi I am there"

    #calling the function
    change_string(string1)

    print("printing the string outside function :",string1)
```

Types of arguments

We can pass different types of arguments at the time of function calling.

- *Required arguments*
- *Keyword arguments*
- *Default arguments*
- *Variable-length arguments*

Required Argument:-

Argument is provided at the time of function calling. As far as the required arguments are concerned, these are the arguments which are required to be passed at the time of function calling with the exact match of their positions in the function call and function definition. If either of the arguments is not provided in the function call, or the position of the arguments is changed, then the python interpreter will show the error.

Example 1

```
#the argument name is the required argument to the function func def
func(name):
   message = "Hi "+name;
   return message;

name = input("Enter the name?")
print(func(name))
```

Output:

```
Enter the name?gauraw
Hi gauraw
```

Example 2

```
#the function simple_interest accepts three arguments and returns the simple interest accordingly def
simple_interest(p,t,r):
        return (p*t*r)/100

p = float(input("Enter the principle amount? ")) r
= float(input("Enter the rate of interest? "))
t = float(input("Enter the time in years? "))
```

```
print("Simple Interest: ",simple_interest(p,r,t))

Output:
Enter the principle amount? 10000 Enter
the rate of interest? 5
Enter the time in years? 2
Simple Interest:  1000.0
```

Example 3

```
#the function calculate returns the sum of two arguments a and b def
calculate(a,b):
   return a+b

calculate(10) # this causes an error as we are missing a required arguments b. Output:
TypeError: calculate() missing 1 required positional argument: 'b'
```

Keyword arguments:-

Python allows us to call the function with the keyword arguments. This kind of function call will enable us to pass the arguments in the random order.
The name of the arguments is treated as the keywords and matched in the function calling and definition. If the same match is found, the values of the arguments are copied in the function definition.

Example 1

```
#function func is called with the name and message as the keyword arguments def
func(name,message):
        print("printing the message with",name,"and ",message)
func(name = "John",message="hello")

Output:
printing the message with John and  hello
```

Example 2

providing the values in different order at the calling

```
#The function simple_interest(p, t, r) is called with the keyword arguments the order of arguments doesn't matter
in this case
def simple_interest(p,t,r):
   return (p*t*r)/100
```

print("Simple Interest: ",simple_interest(t=10,r=10,p=1900))

Output:
Simple Interest: 1900.0

If we provide the different name of arguments at the time of function call, an error will be thrown.

Example 3

```
#The function simple_interest(p, t, r) is called with the keyword arguments. def
simple_interest(p,t,r):
   return (p*t*r)/100

print("Simple Interest: ",simple_interest(time=10,rate=10,principle=1900))      # doesn't find the exact
match of the name of the arguments (keywords)
```

Output:

TypeError: simple_interest() got an unexpected keyword argument 'time'

The python allows us to provide the mix of the required arguments and keyword arguments at the time of function call. However, the required argument must not be given after the keyword argument, i.e., once the keyword argument is encountered in the function call, the following arguments must also be the keyword arguments.

Example 4

```
def func(name1,message,name2):
   print("printing the message with",name1,",",message,",and",name2)
func("John",message="hello",name2="David") #the first argument is not the keyword argument
```

Output:
printing the message with John , hello ,and David

The following example will cause an error due to an in-proper mix of keyword and required arguments being passed in the function call.

Example 5

```
def func(name1,message,name2):
   print("printing the message with",name1,",",message,",and",name2) func("John",message="hello","David")
```

Output:
SyntaxError: positional argument follows keyword argument

Default Arguments:-

Python allows us to initialize the arguments at the function definition. If the value of any of the argument is not provided at the time of function call, then that argument can be initialized with the value given in the definition even if the argument is not specified at the function call.

Example 1

```
def printme(name,age=22):
   print("My name is",name,"and age is",age)
printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function
```

Output:

My name is john and age is 22

Example 2

```
def printme(name,age=22):
   print("My name is",name,"and age is",age)
printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function
printme(age = 10,name="David") #the value of age is overwritten here, 10 will be printed as age
```

Output:

My name is john and age is 22 My name is David and age is 10

Variable length Arguments:-

Sometimes we may not know the number of arguments to be passed in advance. In such cases, Python provides us the flexibility to provide the comma separated values which are internally treated as tuples at the function call.

However, at the function definition, we have to define the variable with * (star) as *<variable - name >.

Example

```
def printme(*names):
   print("type of passed argument is ",type(names)) print("printing the
   passed arguments...")
   for name in names:
      print(name)
```

```
printme("john","David","smith","nick")

Output:
type of passed argument is  <class 'tuple'>
printing the passed arguments...
john
David
smith
```

Scope of variables:-

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

Global variables
Local variables

The variable defined outside any function is known to have a global scope whereas the variable defined inside a function is known to have a local scope.

Example 1

```
def print_message():
   message = "hello !! I am going to print a message." # the variable message is local to the function itself
   print(message)
print_message()
print(message) # this will cause an error since a local variable cannot be accessible here. Output:

hello !! I am going to print a message.
  File "/root/PycharmProjects/PythonTest/Test1.py", line 5, in
   print(message)
NameError: name 'message' is not defined
```

Example 2

```
def calculate(*args):
   sum=0
   for arg in args: sum
      = sum +arg
   print("The sum is",sum)
sum=0
calculate(10,20,30) #60 will be printed as the sum
print("Value of sum outside the function:",sum) # 0 will be printed
```

Output:

The sum is 60
Value of sum outside the function: 0

The Anonymous function (lambda):-

- ***The anonymous function contains a small piece of code***
- ***These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.***
- ***Lambda forms can take any number of arguments.***
- ***Return just one value in the form of an expression.***
- ***They cannot contain commands or multiple expressions.***
- ***An anonymous function cannot be a direct call to print because lambda requires an expression***
- ***Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.***
- ***Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.***

Syntax:-

lambda arguments : expression x

E.g.

```
= lambda a:a+10
print("sum = ",x(20))
```

```
x = lambda a,b:a+b
print("sum = ",x(20,10))
```

Why to use lambda

When we want to execute some task with in function with some input which will be processing parameter passed to function.
E.g.

```
def myfunc(n):
  return lambda a : a * n

x = myfunc(2)          // will execute first fun with parameter 2 then it passed on to lambda with x i.e. 11
print(x(11))
```

Python modules

- *A module allows you to logically organize your Python code.*
- *Grouping related code into a module makes the code easier to understand and use.*
- *A module is a Python object with arbitrarily named attributes that you can bind and reference.*
- *Module is a file consisting of Python code.*
- *Module can define functions, classes and variables. A module can also include runnable code.*

Import statement

- *The import statement is used to import all the functionality of one module into another*

- *We can use the functionality of any python source file by importing that file as the module into another python source file.*
- *We can import multiple modules with a single import statement.*

Syntax:-

import module1,module2,.....module n
ex. import numpy

From-import statement

- *Instead of importing the whole module into the namespace, python provides the flexibility to import only the specific attributes of a module.*
- *This can be done by using from? import statement. The syntax to use the from-import statement is given below.*

Syntax:-
from < module-name> import <name 1>, <name 2>..,<name n>

calculation.py:

```
#place the code in the calculation.py def
summation(a,b):
   return  a+b
def multiplication(a,b):
   return a*b;
def  divide(a,b):
```

```
    return a/b;
```

Main.py:

```
from calculation import summation
#it will import only the summation() from calculation.py
a = int(input("Enter the first number"))
b = int(input("Enter the second number"))
print("Sum = ",summation(a,b))
```

Renaming import file

```
import <module-name> as <specific-name>
```

Python packages

The packages in python facilitate the developer with the application development environment by providing a hierarchical directory structure where a package contains sub-packages, modules, and sub-modules. The packages are used to categorize the application level code efficiently.

4.1 Strings and Operations

Q 1. What is String? With the help of example explain how we can create string variable in python.

Ans:

- Strings data type is sequence of characters, where characters could be letter, digit, whitespace or any other symbol.

a. ***Creation of Strings:***

- Strings in Python can be created using single quotes or double quotes or even triple quotes.
- Example:

```
string1 = 'Welcome'          # Creating a String with single Quotes
string2 = "Welcome"          # Creating a String with double Quotes
string3 = '''Welcome'''      # Creating a String with Triple Quotes
```

b. ***Accessing strings:***

- In Python, individual characters of a String can be accessed by using the method of Indexing, or range slice method [].

- Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

String	**W**	**E**	**L**	**C**	**O**	**M**	**E**
Indexing	0	1	2	3	4	5	6
Negative Index	-7	-6	-5	-4	-3	-2	-1

- ***Example:***

```
string = 'Welcome'
print(string[0])          #Accessing string with index
print(string[1])
print(string[2])
print(srting[0:2])        #Accessing string with range slice method
```

Output:

```
w
e
l
wel
```

c. *Deleting/Updating from a String:*

- In Python, updating or deletion of characters from a String is not allowed as Strings are immutable.
- Although deletion of entire String is possible with the use of a built-in **del keyword**.
- Example:

```
string='welcome'
del string
```

Q 2. Explain Operations on string. Ans:

Operation	Description	Example	Output
Concatenation(+)	-It joins two strings and returns new list.	x="Good" y="Morning" z=x+y print(z)	Good Morning

Append (+=)	-Append operation adds one string at the end of another string	x="Good" y="Morning" x+=y print(x)	Good Morning
Repetition(*)	-It repeats elements from the strings n number of times	x="Hello" y=x*2 print(y)	HelloHello
Slice []	- It will give you character from a specified index.	x="Hello" print(x[1])	e
Range slice[:]	-It will give you characters from specified range slice.	x="Hello" print(x[0:2])	He

4.2 Strings are immutable

Q 3. Python strings are immutable. Comment on this.

Ans:

- Python Strings are immutable, which means that once it is created it cannot be changed.
- Whenever you try to change/modify an existing string, a new string is created.
- As every object (variable) is stored at some address in computer memory.
- The **id()** function is available in python which returns the address of object(variable) in memory. With the help of memory locations/address we can see that for every modification, string get new address in memory.
- Here is the example to demonstration the address change of string after modification.

```
# prints string1 and its address
string1="Good"
print("String1 value is: ",string1)
print("Address of string1 is: ",id(string1)
```

```
# prints string2 and its address
string2="Morning"
print("String2 value is: ",string2)
print("Address of string2 is: ",id(string2)

#appending string1 to string2
string1+= string2
print("String1 value is: ",string1)
print("Address of string1 is: ",id(string1)
```

Output:

```
String1 value is: Good Address
of String1 is: 1000

String2 value is: Morning
Address of String1 is: 2000

String1 value is: GoodMorning
Address of String1 is: 3000
```


- From the above output you can see string1 has address 1000 before modification. In later output you can see that string1 has new address 3000 after modification.
- It is very clear that, after some operations on a string new string get created and it has new memory location. This is because strings are unchangeable/ immutable in nature. Modifications are not allowed on string but new string can be created at new address by adding/appending new string.

4.3 Strings formatting operator

Q 4. Explain various ways of string formatting with example. Ans:

- In python, % sign is a string formatting operator.

- The % operator takes a format string on the left and the corresponding values in a tuple on the right.
- The format operator, % allows users to replace parts of string with the data stored in variables.
- The syntax for string formatting operation is:

"<format>" % (<values>)

- The statement begins with a *format string* consisting of a sequence of characters and *conversion specification*.
- Following the format string is a % sign and then a set of values, one per conversion specification, separated by commas and enclosed in parenthesis.
- If there is single value then parenthesis is optional.
- Following is the list of format characters used for printing different types of data:

Format Symbol	Purpose
%c	Character
%d or %i	Signed decimal integer
%s	String
%u	Unsigned decimal integer
%o	Octal integer
%x or %X	Hexadecimal integer
%e or %E	Exponential notation
%f	Floating point number
%g or %G	Short numbers in floating point or exponential notation

Example: Program to use format sequences while printing a string.

```
name="Amar"
age=8
print("Name = %s and Age = %d" %(name,age))
print("Name = %s and Age = %d" %("Ajit",6))
```

Output:

Name = Amar and Age = 8 Name

= Ajit and Age = 6

In the output, we can see that %s has been replaced by a string and %d has been replaced by an integer value.

4.4 Built-in String methods and functions

Q 5. List and explain any 5 string methods. Or

Q. Explain the use of _____( ) with the help of an example.

Ans.

Sr. No.	Function	Usage	Example
1	capitalize()	This function is used to capitalize first letter of string.	str="hello" print(str.capitalize()) output: Hello
2	isalnum()	Returns true if string has at least 1 character and every character is either a number or an alphabet and False otherwise.	message="JamesBond007" print(message.isalnum()) output: True
3	isalpha()	Returns true if string has at least 1 character and every character is an alphabet and False otherwise.	message="JamesBond007" print(message.isalpha()) output: False
4	isdigit()	Returns true if string has at least 1 character and every character is a digit and False otherwise.	message="007" print(message.isdigit()) output: True
5	islower()	Returns true if string has at least 1	message="Hello"

		character and every character is a lowercase alphabet and False otherwise.	print(message.islower()) output: False
6	isspace()	Returns true if string contains only white space character and False otherwise.	message=" " print(message.isspace()) output: True
7	isupper()	Returns true if string has at least 1 character and every character is an uppercase alphabet and False otherwise.	message="HELLO" print(message.isupper()) output: True
8	len(string)	Returns length of the string.	str="Hello" print(len(str)) output: 5
9	zfill(width)	Returns string left padded with zeros to a total of width characters. It is used with numbers and also retains its sign (+ or -).	str="1234" print(str.zfill(10)) output: 0000001234
10	lower()	Converts all characters in the string into lowercase.	str="Hello" print(str.lower()) output: hello
11	upper()	Converts all characters in the string into uppercase.	str="Hello" print(str.upper()) output: HELLO
12	lstrip()	Removes all leading white space in string.	str=" Hello" print(str.lstrip()) output: Hello

13	rstrip()	Removes all trailing white space in string.	str=" Hello " print(str.rstrip()) output: Hello
14	strip()	Removes all leading white space and trailing white space in string.	str=" Hello " print(str.strip()) output: Hello
15	max(str)	Returns the highest alphabetical character (having highest ASCII value) from the string str.	str="hello friendz" print(max(str)) output: z
16	min(str)	Returns the lowest alphabetical character (having lowest ASCII value) from the string str.	str="hellofriendz" print(min(str)) output: d
17	replace(old,new[, max])	Replaces all or max (if given) occurrences of old in string with new.	str="hello hello hello" print(str.replace("he","Fo")) output: Follo Follo Follo
18	title()	Returns string in title case.	str="The world is beautiful" print(str.title()) output: The World Is Beautiful
19	swapcase()	Toggles the case of every character (uppercase character becomes lowercase and vice versa).	str="The World Is Beautiful" print(str.swapcase()) output: tHE wORLD iS bEAUTIFUL
20	split(delim)	Returns a list of substrings	str="abc,def, ghi,jkl"

		separated by the specified delimiter. If no delimiter is specified then by default it splits strings on all whitespace characters.	print(str.split(',')) output: ['abc', 'def', ' ghi', 'jkl']
21	join(list	It is just the opposite of split. The function joins a list of strings using delimiter with which the function is invoked.	print('-'.join(['abc', 'def', ' ghi', 'jkl'])) output: abc-def- ghi-jkl
22	isidentifier()	Returns true if the string is a valid identifier.	str="Hello" print(str.isidentifier()) output: True
23	enumerate(str)	Returns an enumerate object that lists the index and value of all the characters in the string as pairs.	str="Hello World" print(list(enumerate(str))) output: [(0, 'H'), (1, 'e'), (2, 'l'), (3, 'l'), (4, 'o'), (5, ' '), (6, 'W'), (7, 'o'), (8, 'r'), (9, 'l'), (10, 'd')]

4.5Slice operation

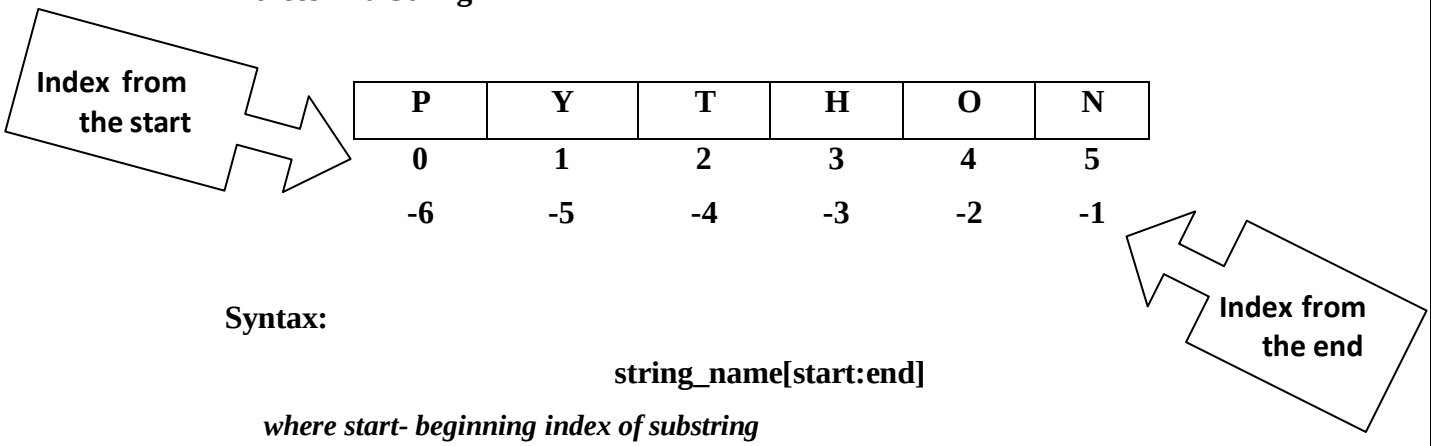
Q 6. What is slice operation? Explain with example. Ans.

Slice: A substring of a string is called a slice.

A slice operation is used to refer to sub-parts of sequences and strings.

Slicing Operator: A subset of a string from the original string by using [] operator known as Slicing Operator.

Indices in a String



Syntax:

string_name[start:end]

where start- beginning index of substring

end -1 is the index of last character

Program to demonstrate slice operation on string objects

```
str="PYTHON"
print("str[1:5]= ", str[1:5])       #characters start at index 1 and extending upto  index 4 #
                                    but not including index 5
print("str[ :6]= ", str[ :6])       # By defaults indices start  at index 0
print("str[1: ]= ", str[1: ])       # By defaults indices ends  upto last index
print("str[ : ]= ", str[ : ])       # By defaults indices start at index 0 and end upto last
                                    #character in the string
#negative index
print("str[-1]= ", str[ -1])        # -1 indicates last character
print("str[ :-2 ]= ", str[ : -2])   #all characters upto -3
print("str[ -2: ]= ", str[ -2: ])   #characters from index -2
print("str[-5 :-2 ]= ", str[ -5: -2])   # characters from index -5 upto character index -3
```

```
OUTPUT str[1:5]=
YTHO str[ :6]=
PYTHON str[1: ]=
YTHON str[ : ]=
PYTHON
```

str[-1]= N

str[:-2]= PYTH

str[-2:]= ON

str[-5 :-2]= YTH

Specifying Stride while Slicing Strings

- In the slice operation, you can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string.
- The default value of stride is 1, i.e. where value of stride is not specified, its default value of 1 is used which means that every character between two index number is retrieved.

Program to use slice operation with stride

```
str=" Welcome to the world of Python"
print("str[ 2: 10]= ", str[2:10])          #default stride is 1
print("str[ 2:10:1 ]= ", str[2:10:1])      #same as stride=1
print("str[ 2:10:2 ]= ", str[2:10:2])      #skips every alternate character
print("str[ 2:10:4 ]= ", str[2:10:4])      #skips every fourth character
```

OUTPUT

str[2: 10]=lcome to

str[2: 10]= lcome to

str[2:10:2]=loet

str[2:10:4]=l

- Whitespace characters are skipped as they are also part of the string.
-

4.6ord() and chr() functions

Q 7.Write a short note on ord() and chr() functions Ans.

- The ord() function return the ASCII code of the character

- The chr() function returns character represented by a ASCII number.

ch='R' print(ord(ch)) **OUTPUT** 82	print(chr(82)) **OUTPUT** R	print(chr(112)) **OUTPUT** p	print(ord('p')) **OUTPUT** 112

4.7 in and not in operators

Q 8.Write a short note on in and not in operators OR

Q.With the help of example, explain significance of membership operators.

Ans.

- *in* and *not in* operators can be used with strings to determine whether a string is present in another string. Therefore the *in* and *not in* operator is known as membership operators.

- For example:

str1=" Welcome to the world of Python!!!"
str2="the"
if str2 in str1:
print("found")
else:
print("Not found")
 	
OUTPUT
Found
	str1=" This is very good book"
	str2="best"
	if str2 in str1:
	print("found")
	else:
	print("Not found")
	
	OUTPUT
	Not found

- You can also use *in* and *not in* operators to check whether a character is present in a word.
- For example:

'u' in "starts"	'v' not in "success"

OUTPUT	OUTPUT
False	True

4.8 Comparing strings

Q 9. Explain string comparison operator with example? Ans.

- Python allows us to combine strings using relational (or comparison) operators such as >, <, <=,>=, etc.
- Some of these operators along with their description and usage are given as follows:

Operator	Description	Example
==	If two strings are equal, it returns True.	>>>"AbC"=="AbC" True
!= or <>	If two strings are not equal, it returns True.	>>>"AbC"!="Abc" True
>	If the first string is greater than the second, it returns True.	>>>"abc">"Abc" True
<	If the second string is greater than the first, it returns True.	>>>"abC"<"abc" True
>=	If the first string is greater than or equal to the second, it returns True.	>>>"aBC">=""ABC" True
<=	If the second string is greater than or equal to the first, it returns True.	>>>"ABc"<="ABc" True

- These operators compare the strings by using ASCII value of the characters.
- The ASCII values of A-Z are 65-90 and ASCII code for a-z is 97-122.
- For example, book is greater than Book because the ASCII value of 'b' is 98 and 'B' ***is 66.***

String Comparison Programming Examples: (Any one)

- There are different ways of comparing two strings in Python programs:

- Using the **==(equal to)** operator for comparing two strings:
  - If we simply require comparing the values of two variables then you may use ***the '==' operator.***
  - If strings are same, it evaluates to True, otherwise False.

  - Example1:

    first_str='Kunal works at Phoenix'
    second_str='Kunal works at Phoenix'
    print("First String:", first_str) print("Second String:", second_str) #comparing by ==
    if first_str==second_str:
        print("Both Strings are Same")
    else:
        print("Both Strings are Different")

    Output:
    First String: Kunal works at Phoenix Second String: Kunal works at Phoenix Both Strings are Same

  - Example2(Checking Case Sensitivity):

    first_str='Kunal works at PHOENIX'
    second_str='Kunal works at Phoenix'
    print("First String:", first_str)
    print("Second String:", second_str)
    #comparing by ==
    if first_str==second_str:
        print("Both Strings are Same")
    else:
        print("Both Strings are Different")

Output:

First String: Kunal works at PHOENIX

Second String: Kunal works at Phoenix

Both Strings are Different

- Using the **!=(not equal to)** operator for comparing two strings:
  - The != operator works exactly opposite to ==, that is it returns true is both the strings are not equal.
  - Example:

```
first_str='Kunal works at Phoenix'
second_str='Kunal works at Phoenix'
print("First String:", first_str) print("Second
String:", second_str) #comparing by !=
if first_str!=second_str:
        print("Both Strings are Different")
else:
        print("Both Strings are Same")
```

output:

First String: Kunal works at Phoenix Second String: Kunal works at Phoenix Both Strings are Same

- Using the **is** operator for comparing two strings:
  - The **is** operator compares two variables based on the object id and returns True if the two variables refer to the same object.
  - Example:

```
name1=”Kunal”
name2=”Shreya”
print(“name1:”,name1)
```

print("name2:",name2)

print("Both are same",name1 is name2)

name2="Kunal" print("name1:",name1)

print("name2:",name2)

print("Both are same",name1 is name2)

- Output:

  name1=Kunal

  name2=Shreya

  Both are same False

  name1=Kunal

  name2=Kunal

  Both are same True

- In the above example, name2 gets the value of Kunal and subsequently name1 and name2 refer to the same object.
-

4.9 Iterating strings

Q. No.10 How to iterate a string using:

Ans.

i) for loop with example

ii) while loop with example

Ans.

- String is a sequence type (sequence of characters).
- We can iterate through the string using:

i) for loop:

- for loop executes for every character in str.
- The loop starts with the first character and automatically ends when the last character is accessed.
- Example-

str="Welcome to python"

for i in str:

print(i,end=' ')

Output-

W e l c o m e t o P y t h o n

ii) while loop:

- We can also iterate through the string using while loop by writing the following code.
- Example-

message=" Welcome to python"

index=0

while index < len(message):

letter=message[index]

print(letter,end=' ')

index=index+1

Output-

W e l c o m e t o P y t h o n

- In the above program the loop traverses the string and displays each letter.
- The loop condition is index < len(message), so the moment index becomes equal to the length of the string, the condition evaluates to False, and the body of the loop is not executed.
- Index of the last character is len(message)-1.

4.10 The string module

Q. No. 11 Write a note on string module?

- The string module consists of a number of useful constants, classes and functions.
- These functions are used to manipulate strings.

- String Constants: Some constants defined in the string module are:
    - string.ascii_letters: Combination of ascii_lowecase and ascii_uppercase constants.
    - string.ascii_lowercase: Refers to all lowercase letters from a-z.
    - string.ascii_uppercase: Refers to all uppercase letters from A-Z.
    - string.lowercase: A string that has all the characters that are considered lowercase letters.
    - string.uppercase: A string that has all the characters that are considered uppercase letters.
    - string.digits:Refers to digits from 0-9.
    - string.hexdigits: Refers to hexadecimal digits,0-9,a-f, and A-F.
    - string.octdigits: Refers to octal digits from 0-7.
    - string.punctuation: String of ASCII characters that are considered to be punctuation characters.
    - string.printable: String of printable characters which includes digits, letters, punctuation, and whitespaces.
    - string.whitespace: A string that has all characters that are considered whitespaces like space, tab, return, and vertical tab.
- Example: (Program that uses different methods such as upper, lower, split, join, count, replace, and find on string object)

```
str="Welcome to the world of Python"
print("Uppercase-", str.upper())
print("Lowercase-", str.lower())
print("Split-", str.split())
print("Join-", '-'.join(str.split()))
print("Replace-",str.replace("Python","Java"))
print("Count of o-", str.count('o'))
print("Find of-",str.find("of"))
```

JOIN BIGGEST SPPU ENGINEERING GROUP

1

PuneEngineers Channel

https://whatsapp.com/channel/0029Va7akvF8V0theLGDwI21

2

PuneEngineers Channel

https://t.me/PuneEngineers

3

SPPU Engineering Group

https://t.me/SPPUBTECH

OVER 30,000+

PUNE ENGINEERS

SPPU ENGINEERING @SPPUBTECH

JOIN NOW