FUNCTIONS

INTRODUCTION: Functions are common to all programming languages, and it can be defined as a block of re-usable code to perform specific tasks. But defining functions in Python means knowing both types first: built-in and user-defined. Built-in functions are usually a part of Python packages and libraries, whereas user-defined functions are written by the developers to meet certain requirements. In Python, all functions are treated as objects, so it is more flexible compared to other high-level languages.

DEFINING A FUNCTION: 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 .The code block within every function 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(parameters):
statement(s)
statement(s)
statement(s)
return()

Example: Def greet(name):
"""This function greets to
the person passed in as
Parameter"""
print("Hello, " + name + ". Good morning!")

CALLING A FUNCTION: Once we have defined a function, we can call it from another function, program or even the Python prompt. To call a function we simply type the function name with appropriate parameters.

Syntax: function_name (parameters)

Example: Def greet(name):
"""This function greets to
the person passed in as
Parameter"""
print("Hello, " + name + ". Good morning!")

greet('Paul)#function_call

FUNCTION ARGUMENTS: In Python, user-defined functions can take four different types of arguments. The argument types and their meanings, however, are pre-defined and can't be changed. But we can, instead, follow these pre-defined rules to make their own custom functions.

Default arguments: Default values indicate that the function argument will take that value if no argument value is passed during function call. The default value is assigned by using assignment (=) operator.

E.g: def defaultArg(name,msg="hello"):
print(msg) #hello

defaultArg("hai")

Here we are having two argument but msg="hello" is the default argument , so here while function call we may or may not give value to msg, as it takes hello by default, so it is called as default arguments.

Required arguments: Required arguments are the mandatory arguments of a function. These argument values must be passed in correct number and order during function call.

E.g: def requiredArg(name,msg):
print (msg)

requiredArg("radha","hai")

Here we did not assign any value to any of the arguments so while function call we need to assign value in the same order,so that "radha" will be stored in name and "hai " will be stored in msg

Keyword arguments: Keyword arguments are relevant for Python function calls. The keywords are mentioned during the function call along with their corresponding values. These keywords are mapped with the function arguments so the function can easily identify the corresponding values even if the order is not maintained during the function call.

E.g: def keywordArg(name,msg):
print (msg,name)

keywordArg(msg="radha",name="hai")

In keyword arguments we may change the order of assigning values by using keyword of the aruguments.

Variable number of arguments: This is very useful when we do not know the exact number of arguments that will be passed to a function. Or we can have a design where any number of arguments can be passed based on the requirement.

E.g: def variableArg(*varargs):
print(varargs)

variableArg(1,2,2,3,4)

ANONYMUS FUNCTION: n Python, anonymous function is a function that is defined without a name. While normal functions are defined using the def() keyword, in Python anonymous functions are defined using the lambda keyword. Lambda functions are mainly used in combination with the functions filter(), map() and reduce().

Syntax: lambda arguments: expressions
E.g: f=lambda x, y : x
f(1,1) # 2

GLOBAL AND LOCAL VARIABLES: Global variables are the one that are defined and declared outside a function and we need to use them inside a function.

E.g: def abc():
print(a)

a=”hello”
abc()

- Here we declared variable ‘a’ globally before the function call so value of ‘a’ will be taken as hello globally, this is called global declaration.

- If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

E.g: def abc():
a=”hai”
print(a)

a=”hello”
abc()

- Here in the above example we had declared ‘a’ globally and locally. So in the function it will assume’ a ‘ as’ hai’ and outside the function ‘a ‘ is assumed as ‘hello’

- If we want to declare a variable inside the function and want to use it globally then “global” keyword is used.

E.g: def f():

```
    global s
    print s    #"Python is great!"
    s = "Python is a language"
    print s   #"Python is a language"

s = "Python is great!"
f()
print s   #"Python is a language"
```

- Here initially s is declared as “Python is great!" , later f() function call is triggered and “s “will be globally declared by global keyword. Next s is again initialized, here this “s” overwrites the value and stores "Python is a language” and prints the same