Class – 12 Computer Science

Term – I (2021-22)

TERM 1:

Unit I: Computational Thinking and Programming – 2

- Revision of Python topics covered in Class XI.
- Functions: types of function (built-in functions, functions defined in module, user defined functions), creating user defined function, arguments and parameters, default parameters, positional parameters, function returning value(s), flow of execution, scope of a variable (global scope, local scope)
- Introduction to files, types of files (Text file, Binary file, CSV file), relative and absolute paths
- Text file: opening a text file, text file open modes (r, r+, w, w+, a, a+), closing a text file, opening a file using with clause, writing/appending data to a text file using write() and writelines(), reading from a text file using read(), readline() and readlines(), seek and tell methods, manipulation of data in a text file
- Binary file: basic operations on a binary file: open using file open modes (rb, rb+, wb, wb+, ab, ab+), close a binary file, import pickle module, dump() and load() method, read, write/create, search, append and update operations in a binary file
- CSV file: import csv module, open / close csv file, write into a csv file using csv.writerow() and read from a csv file using csv.reader( )

Working with Function

Function: - A function is a subprogram that act on data and often return a value.

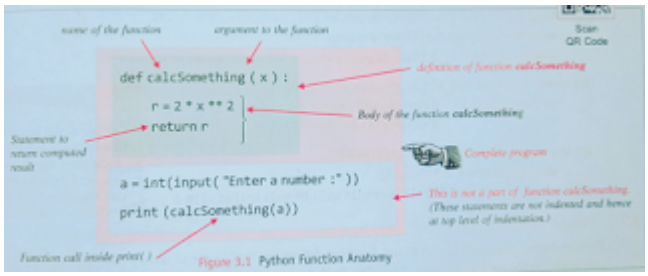
Python function types:-

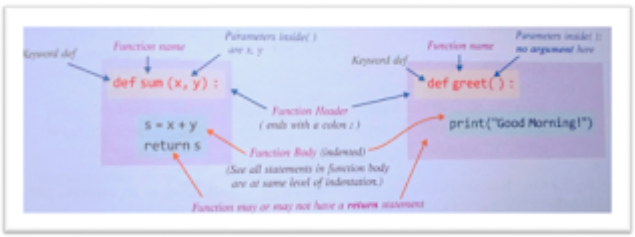
1 = Built in function: - These are pre-define function and always available for use. You have used some of them like - len (), type (), int (), input () etc.

2 = Function defined in modules: - These functions are pre-defined in particular models and can only be used when the corresponding model is imported.
For example: - If we want to find the square root of any number then we have import math module then call the function - sqrt ()

3 = User defined functions: - These are define by the programmer. As programmer you can create your own function.

Defining function in python:-
Look figure carefully --





Function header: - The first line of the function definition that beings with keyword Def and ends with a colon (:), specifies the name of the function and its parameters.

Parameters: - Variables that are listed within the parentheses of a function header.

Function body: - The block of statement/indented - statement beneath function header that defines the action performed by the function.

Indentation: - The blank space in the beginning of statement within a block. All statements within same block have same indentation.

Flow of execution: - The flow of execution refers to the order in which statement are executed during a program run.

For example: -

```
def calcSum (x,y):
        s = x + y
        return s
num1 = float (input ("Enter the first number: "))
num2 = float (input("Enter the second number : "))
sum = calSum (num1,num2)
print("Sum of two given number is ",sum)
```

Argument: - The values being passed through a function call statement are called argument (or actual parameters or actual argument).
For example:-

```
def calcSum ( x , y ):
        s = x + y
        return s

print (calcSum ( 2 , 3 ))
a = 5
b = 6
print (calcSum ( a , b ))
d = 10
print (calcSum ( 9 , d ))
```

- Here a , b , d , 2 , 3 , 9 are "arguments" which is used in call function.

Parameters: - The values received in the function definition header are called parameter (or formal parameters or formal arguments).
For example: -

```
def calcSum ( x , y ):
        :
```

- Here x , y are "parameters"

Passing parameters:-

Python support three types of formal arguments/parameters:

1:- Positional argument (required arguments): - When the functions call statement must match the number and order of arguments as define in the functions definition this is called the position argument matching.
For example:-

```
def check (a,b,c):
      :
```

Then possible functions call for this can be:-

```
check ( x , y , z ) # 3 values( all variables) passed

check ( 2 , x , y ) # 3 values ( literal + variables ) passed

check ( 2 , 3 , 4 ) # 3 values ( all literal ) passed
```

Thus through such functions calls -
• *The argument must be provided for all parameters (required)*
• *The values of argument are matched with parameters, position (order) wise (positional)*

2:- Default arguments: - A parameter having defined value in the function header is known as a default parameter.
For example:-

```
def interest( principal , time , rate = 10 ) :
            :
```

If:-

```
si = interest ( 5400,2 ) #third argument missing
```

So the parameter principal get value 5400, time get 2 and since the third argument rate is missing, so default value 0.10 is used for rate.

If:-
si = interest (6100 ,3 ,0.15) # no argument missing

So the parameter principal get value 6100, time get 3 and the parameter rate gets value 0.15.

• That means the default values (values assigned in function header) are considered only if no value is provided for that parameter in the function call statement.
• Default argument are useful in situations where some parameters always have same value.

You can understand more by seeing below examples:-

```
def interest ( prin , time , rate = 0.10) # legal

def interest ( prin , time = 2 , rate) # illegal ( default parameter before required parameter )

def interest ( prin = 2000 ,time = 2 ,rate) # illegal
# (same reason as above)

def interest ( prin , time = 2 , rate = 0.10 ) # legal

def interest ( prin = 2000 , time = 2 , rate = 0.10) # legal
```

Some advantages of the default parameters are listed below:-
• They can be used to add new parameters to the existing functions.
• They can used to combine similar function into one.

3:- Keyword (*or named*) arguments:-
Keyword arguments are the named arguments with assigned values being passed in the function call statement.
For example:-

```
def interest ( prin , time , rate ) :
      return prin * time * rate

print (interest ( prin = 2000 , time = 2 , rate 0.10 ))

print (interest ( time = 4 , prin = 2600 , rate = 0.09 ))

print (interest ( time = 2  , rate = 0.12  , prin = 2000 ))
```

• All the above functions call are valid now, even if the order of arguments does not match.

Using multiple argument type together:-

Python allows you to combine multiple argument types in a function call.

Rules for combining all three types of arguments:-

• And argument list must first contain positional (required) arguments followed by any keyword argument.
• Keyword arguments should be taken from the required arguments preferably.
• You cannot specify a value for an argument more than once.
For example:-

```
def interest ( prin , cc , time = 2 , rate = 0.09 ):
     return prin * time * rate
```

Function call statement	Legal / Illegal	Reason
`interest(prin = 3000, cc = 5)`	legal	non-default values provided as named arguments
`interest(rate = 0.12, prin = 5000, cc = 4)`	legal	keyword arguments can be used in any order and for the argument skipped, there is a default value
`interest(cc = 4, rate = 0.12, prin = 5000)`	legal	with keyword arguments, we can give values in any order
`interest(5000, 3, rate = 0.05)`	legal	positional arguments before keyword argument; for skipped argument there is a default value
`interest(rate = 0.05, 5000, 3 )`	illegal	keyword argument before positional arguments
`interest(5000, prin = 300, cc =2)`	illegal	Multiple values provided for **prin** ; once as positional argument and again as keyword argument
`interest(5000, principal = 300, cc = 2 )`	illegal	undefined name used (**principal** is not a parameter)
`interest(500, time = 2, rate = 0.05 )`	illegal	A required argument (**cc**) is missing.