**2.16 Python Functions**

Large programmes need to be divided into small logical units. A ***function*** is a small, isolated unit of code with a name, and it can perform a definite work. From any part of the programme, we can call this function by name. Then the computer will execute that code to do the defined job. When we are calling a function, we must pass some data to the function, for its proper working. These data variables are called ***arguments***. On getting the arguments, the function will execute the instructions for doing the prescribed work. After doing it, control will return to the main programme with one or more answers. Functions are divided into ***user defined functions*** or ***built-in functions*** or ***library functions.***

**Built-in functions**

One of the major advantages of Python is the availability of various libraries for various applications like scientific computation, graphics, networking etc. The standard library is divided into many modules like time, random, pickle, system etc. Each module consists of many built-in functions.

To get the service of such functions, we must instruct the compiler to load the corresponding module or function library to RAM. Following are the examples

*from math import sin*

*print(sin(x))*

Here we are loading the function to calculate sin(x) from the module named math. But with this type of loading, we can calculate only sin(x) using the math library. Other functions defined in the math library will not work. The same thing can do in another way.

*from math import \**

*print(sin(x), cos(x))*

In this example, we are importing the entire mathematical library into RAM. So, we can call any functions defined in that module. There is another method to do the same thing. But there may be an overloading of RAM.

*import math*

*print(math.sin(x))*

*print(math.cos(x))*

If the library name is large, the same instructions can be in another form.

*import math as m*

*print(m.sin(x))*

*print(m.cos(x))*

Similarly, there are many functions defined in different libraries.

*from NumPy import \**

*from pylab import \**

**List of important mathematical functions**

**(**This is a partial list. A full list is available on the web. To activate these functions, we must import math library**)**

**ceil(*x*)**: Function returns the smallest integer value greater than or equal to *x*. Input x must be float.

*x=12.8*

*y=12.3*

*ceil(x)=13*

*ceil(y)=13*

**floor(*x*)**: Function returns the smallest integer value less than or equal to *x*. Input x must be float.

*x=12.8*

*y=12.3*

*floor(x)=12*

*floor(y)=12*

**fabs(*x*)**: Function returns the absolute value of *x* as a float. Input x may be integer or float.

**fmod(*x*, *y*)**: Function returns the reminder in x/y as a float. Inputs may be integer or float. It is almost identical to x%y. But there may be a slight difference in the accuracy. According to the developing team of Python fmod() is more accurate when working with floats, while x%y is more accurate when working with *integers.*

*x=10*

*y=9*

*fmod(x,y)=1.0*

**fsum(*x*)**: Function returns a floating point sum of values in the list named x

*x=[.1,.2,.3,.4]*

*fsum(x)=1.0*

**trunc(*x*)**: Function returns the integer part of the float x after truncation.

*x=12.473*

*trunc(x)=12*

**exp(*x*)**: Function returns e\*\*x.

**log(*x*, *b*)**: With one argument, function returns the natural logarithm of *x* to the base *e*. With two arguments, it returns the logarithm of *x* to the given base b

**log10(*x*)**: Function returns the logarithm of *x* base-10. This is usually more accurate than log(x, 10).

**pow(*x*, *y*)**: Function returns x raised to the power y.

**sqrt(*x*, *y*)**: Function returns the square root of *x* as a float.

**acos(*x*)**: Function returns the arc cosine of *x*, if x is in radians.

**asin(*x*)**: Function returns the arc sine of *x*, if x is in radians.

**atan(*x*)**: Function returns the arc tangent of *x*, if x is in radians.

**cos(*x*)**: Function returns the cosine of *x,* if x is in radians.

**sin(*x*)**: Function returns the sine of *x,* if x is in radians.

**tan(*x*)**: Function returns the tangent of *x,* if x is in radians.

**cosh(*x*):** Function returns the hyperbolic cosine of *x*, if x is in radians.

**sinh(*x*)**: Function returns the hyperbolic sine of *x*, if x is in radians.

**tanh(*x*)**: Function returns the hyperbolic tangent of *x*, if x is in radians.

**degrees(*x*)**: Function converts angle *x* from radians to degrees.

**radians(*x*)**: Function converts angle *x* from degrees to radians.

**factorial(*x*)**: Function returns factorial of x as an integer. X must be a positive integer.

**List of important general functions**

For the easiness of programming, python supports many general functions defined in general library. No need of any import statements to access these functions within a programme. This is a partial list. Full list is available in web.

**chr(*i*)** : Function returns a string of one character whose ASCII code is the integer

*chr(97)=’a’*

**cmp(*x*, *y*)** : Used to compare the two numbers *x* and *y* and return an integer according to the outcome. The return value is negative if x < y, zero if x == y and positive if x > y.

*x=12*

*y=14*

*cmp(x,y)= -1*

**help(*object*)**: Invoke the built-in help system in interactive mode. If no argument is given, the interactive help system starts on the interpreter console. For example, type ‘help (input)’ in the command prompt. Then the system will show the exact syntax of the instruction as follows.

*input(...)*

*input([prompt]) -> value*

**len(*s*) :** Return the number of items in a list or the number of characters including white space in a string or set.

*x=[2,8,3.7,4]*

*len(x)=4*

*x=’hello’*

*len(s)=5*

**list(x) :**Return a list of elements in x , if x represents a list

*x=[2,8,3.7,4]*

*list(x)*

*[2, 8, 3.7, 4]*

**max(x)**: If x represents a list or set, function will return the element of highest value.

*x=[2,8,3.7,4]*

*max(x) =8*

**long(*x*)** : Convert a string or number to a long integer.

**min(x)** : If x represents a list or set, function will return the element of lowest value.

*x=[2,8,3.7,4]*

*max(x) =2*

**oct(*x*)** : Convert an integer number to an octal string. The result is a valid Python expression.

**hex(x):** Convert an integer number to a hexadecimal number string. The result is a valid Python expression.

**round(*x*, *n*)**:Return the floating-point value *x* rounded to *n* digits after the decimal point. If *n* is omitted, it defaults to zero.

*round(23.458)=23.0*

*round (23.458,1)=23.5*

**min(*n1, n2, n3,...*)** : If n1, n2,n3 ... are variables representing numbers, function will return the value of the lowest. If n1, n2, n3... are numbers, system will show the lowest number.

*x=12*

*y=15*

*min(x,y)=12*

*min(12.8, 25, 2, 14.32)=2*

**max(*n1, n2, n3,...*)**: If n1, n2,n3 ... are variables representing numbers, function will return the value of the highest. If n1, n2, n3... are numbers system will the highest number.

*x=12*

*y=15*

*max(x,y)=15*

*max(12.8, 25, 2, 14.32)=14.32*

**User defined functions**

A user defined function is a named part of the programme made by the user. It can be considered as a sub-programme, within the main programme. The naming rules of function are the same as that of a variable. It can be invoked from the other parts of the programme as often needed. Let us go through an example. Suppose we want to find the square root and cube root of numeral very often in a program. So, we can incorporate two program segments named sqr and cube as functions. At any instant when we need a square root or cube root, we can ask the control to go to the respective function to find it and return. Go through the following examples.

*def cube(x)*

*return x\*x\*x*

*def sqr(x)*

*return x\*x*

*num=input(‘Enter the number’)*

*print (‘The square of the number is ‘, sqr(num))*

*print(‘The cube of the number is ‘, cube(num))*

‘def’ is the keyword to make a function definition. ‘sqr’ and ‘cube’ is the name of the function. The general syntax of a function definition is the following.

*def function name (list of arguments to be passed if any)*

*function body*

*return name of the argument if any*

The rule for function naming is the same as that of variable naming. For the proper working of a function, we must pass the data to the function from the main programme. The variable using in the function definition is called ***dummy argument***. The number of arguments used in the function call must be the same as that of the function title. The variable name used in both places may be different. In the above example, the variable in a function call is ‘num’ and that in function title is ‘x’. If there is no transfer of data from the main programme to function, we can omit the dummy argument. But the parenthesis along with the function name is compulsory. We can write all calculations and instructions just below the function definition statement with an indent. These lines will form the ***function body***. After executing all instructions in the intended block, control will return to the main programme. During returning to the main programme, if any data is to be transferred, we can use the statement ’return’. The syntax is,

*return (list of variables to transfer, separated by comma).*

But, the usage of return() is optional. Go through the following example.

It can be noted that the argument in the function call and the argument in the function title is the same in this example. It is not necessarily to be the same. We can use two different variables for the argument in the function call and function title. But the number of arguments used in the function call and the function title must be the same.