Programming and Problem Solving through Python Language
O Level / A Level

Chapter - 6 : Functions

Approach of Problem Solving

There are three general approaches to writing a program:

1. **Top down** - In the top down approach one starts with the toplevel routine and move down to the low level routine.
2. **Bottom up** - The bottomup approach works in the opposite direction on begins with the specific routines, build them into progressively more complex structures, and end at the top level routine.
3. **Ad hoc** - The ad hoc approach specifies no predetermined method.

Top-down approach

- A top down approach also helps one to clarify the overall structure and operation of the program before one code the low level functions.
- The top down method starts with a general description and works towards specifics.
- A good way to design a program is to define exactly what the program is going to do at the top level.
- Each entry in the list should contain only one functional unit.
- A functional unit can be thought of as a black box that performs a single task. Modular programming
- Modular programming is a style that adds structure and readability to the program code.
- It may not make much difference on small projects, but as one starts to work on something bigger it can make the code much easier to read and maintain.
- Structuring the code is a simple task of splitting the program into manageable part so that each part is self contained.
- By creating these self contained modules, one can focus on programming each part.

Functions

- A function is a named, independent section of Python code that performs a specific task and optionally returns a value to the calling program.
- A function is named. Each function has a unique name.
- By using the name in another part of the program, one can execute the statements contained in the function. **This is known as calling the function**.
- A function can be called from within any other function.
- A function is independent.
- A function can perform its task without interference from or interfering with other parts of the program.

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. We 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 docstring.**
- 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( parameters ) :
  "function_docstring"
      function_local variable
      function statements
      return [expression]
```

Example

```
# Function definition is here
def printme( str ):
  "This prints a passed string into this function"      # docstring
    print (str)
    return

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

Output

```
This is first call to the user defined function!
Again second call to the same function
```

Built-in Python Functions

SNo	Function	Description
1.	input()	Allowing user input
2.	print()	Prints to the standard output device
3.	int()	Returns an integer number
4.	float()	Returns a floating point number
5.	list()	Returns a list
6.	dict()	Returns a dictionary (Array)
7.	set()	Returns a new set object
8.	str()	Returns a string object
9.	tuple()	Returns a tuple
10.	type()	Returns the type of an object
11.	len()	Returns the length of an object
12.	format()	Formats a specified value
13.	abs()	Returns the absolute value of a number
14.	eval()	Evaluates and executes an expression
15.	round()	Rounds a numbers
16.	max()	Returns the largest item in an iterable
17.	min()	Returns the smallest item in an iterable
18.	oct()	Converts a number into an octal
19.	pow()	Returns the value of x to the power of y
20.	range()	Returns a sequence of numbers, starting from 0 and increments by 1 (by default)

1. input() Function

The **input()** function allows user input.

`input(prompt)`

```
x = input('Enter your name:')
```

2. print() Function

- The **print()** function prints the specified message to the screen, or other standard output device.
- The message can be a string, or any other object, the object will be converted into a string before written to the screen

print*(object(s)*, sep=*separator*, end=*end*, file=*file*, flush=*flush*)

object(s)	Any object, and as many as you like. Will be converted to string before printed
sep='*separator*'	Optional. Specify how to separate the objects, if there is more than one. Default is ' '
end='*end*'	Optional. Specify what to print at the end. Default is '\n' (line feed)
file	Optional. An object with a write method. Default is sys.stdout
flush	Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False

```
print("Hello", "how are you?")
x = ("apple", "banana", "cherry")
print(x)
print("Hello", "how are you?", sep="---")
```

3. int() Function

The int() function converts the specified value into an integer number.

Syntax : int(*value*, *base*)

value A number or a string that can be converted into an integer number
base A number representing the number format. Default value: 10

```
x = int("12")
print(x+10)
```

3. float() Function

The float() function converts the specified value into a floating point number..

Syntax : `float(value)`

```
x = float(3)
print(x)
x = float("3.500")
print(x)
```

4. list() Function :

The list() function creates a list object.
A list object is a collection which is ordered and changeable.

5. dict() Function :

The dict() function creates a dictionary.
A dictionary is a collection which is unordered, changeable and indexed.

6. set() Function

The set() function creates a set object.
The items in a set list are unordered, so it will appear in random order.

7. str() Function

The str() function converts the specified value into a string.

8. tuple() Function

The tuple() function creates a tuple object.
We cannot change or remove items in a tuple.

Note – Function list(), dict(), set(), str() & tuple() already covered in Chapter -5 .

9. type() Function

The type() function returns the type of the specified object.

```
a = ('apple', 'banana', 'cherry')
b = "Hello World"
c = 33

x = type(a)
y = type(b)
z = type(c)
```

10. len() Function

The len() function returns the number of items in an object.
When the object is a string, it returns the number of characters in the string.

```
mylist = ["apple", "banana", "cherry"]
x = len(mylist)
print(x)

mylist = "Hello"
x = len(mylist)
print(x)
```

11. format() Function

The format() function formats a specified value into a specified format

format(*value*, *format*)

Parameter	Description
value	A value of any format
format	The format you want to format the value into. Legal values: '<' - Left aligns the result (within the available space) '>' - Right aligns the result (within the available space) '^' - Center aligns the result (within the available space) '=' - Places the sign to the left most position '+' - Use a plus sign to indicate if the result is positive or negative '-' - Use a minus sign for negative values only ' ' - Use a leading space for positive numbers ',' - Use a comma as a thousand separator '_' - Use a underscore as a thousand separator 'b' - Binary format 'c' - Converts the value into the corresponding unicode character 'd' - Decimal format 'e' - Scientific format, with a lower case e 'E' - Scientific format, with an upper case E 'f' - Fix point number format 'F' - Fix point number format, upper case 'g' - General format 'G' - General format (using a upper case E for scientific notations) 'o' - Octal format 'x' - Hex format, lower case 'X' - Hex format, upper case 'n' - Number format '%' - Percentage format

```
x = format(0.5, '%')      Output - 50.000000%
x = format(255, 'x')      Output - ff
```