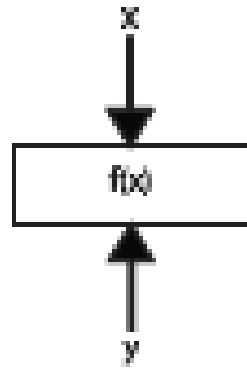


Introduction to Function

Introduction to Functions

- A function is a group of statements that aims to perform a specific task. For example, in mathematics, a mathematical function $f(x)=x^2$ takes the value of x and returns x^2 as output.

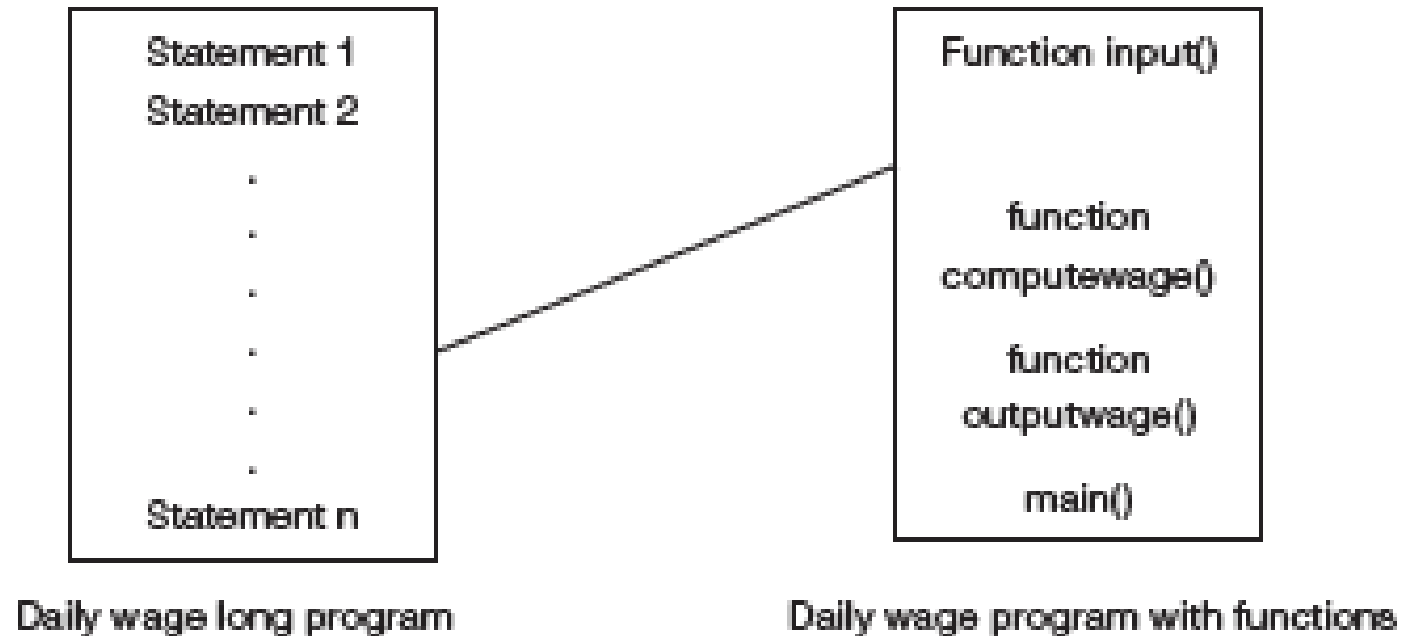


Mathematical Function

- Mathematical Function can be implemented straightway as a function in python. For that, it is necessary to group statements to perform the square function,
- A function is also known as a procedure, routine, or subprogram.
- The advantage of a function is that, instead of writing a long program, one can divide that program into a set of functions.



Illustration for use of Functions

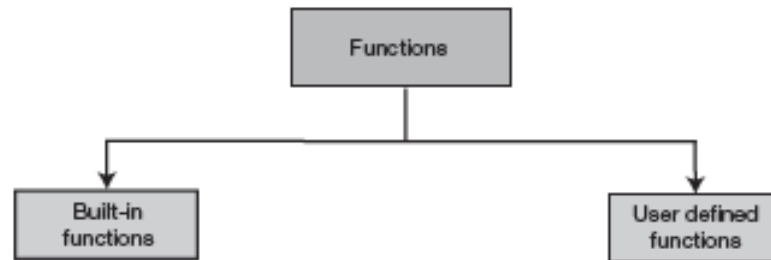


Advantages of the function-oriented approach

No.	Advantages	Descriptions
1.	Avoids repetition	The same code across programs can be replaced by functions.
2.	Reduction of size	The removal of repeating code by function reduces the size of the program.
3.	Memory saving	Saves memory space by using the same set of statements for a different set of data.
4.	Code reuse	Code reuse removes the duplication of work. Once a function is written, it can be reused when there is a need to perform the task.
5.	Modularisation	The decomposition of programs into functions makes design effective.
6.	Simpler code	A code that uses functions makes the program simple and improves its understandability.
7.	Specific tasks	Functions can be specialised.
8.	Parallel development	A project can be divided into many modules, and as a result many programmers can work simultaneously. This improvement in parallelisation fastens the project's development and facilitates teamwork.
9.	Testing	Smaller modules make testing simple. A function can be comprehensively tested and debugged as an isolation of the error, and fixing it becomes simpler.
10.	Portability	Functions can be shared across the programs leading to portability.

Types of Functions

- There are many types of functions.

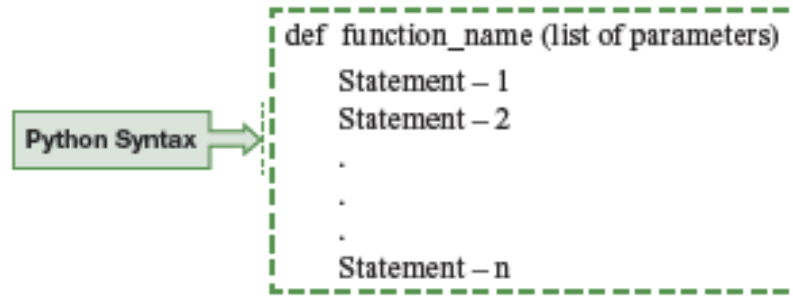


- The following script illustrates some of these built-in functions

```
>>> float(64)
64.0
>>> str(43)
'43'
```

Anatomy of a User-defined Function

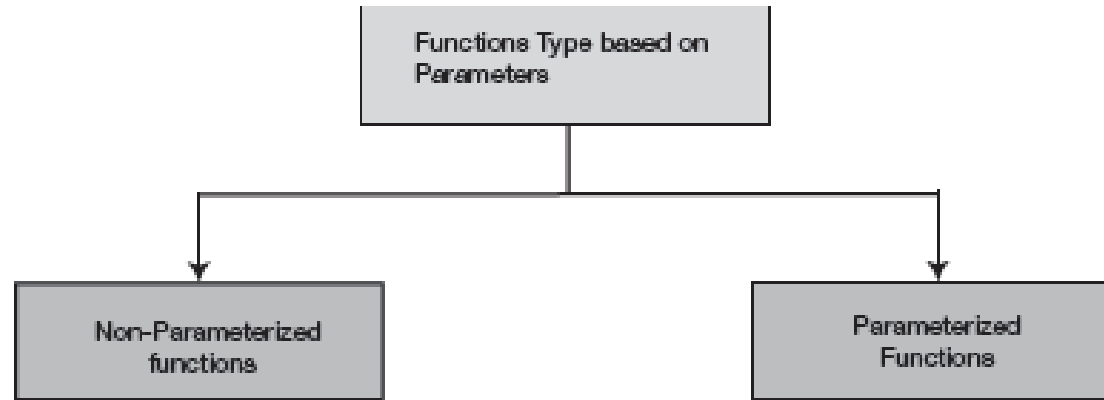
- The syntax of the user-defined function is given below,



```
def function_name (list of parameters)
    Statement - 1
    Statement - 2
    .
    .
    .
    Statement - n
```

- The Components of a function are given below,
 1. Function definition
 2. Function call

Types of Functions Based on Parameters



Non-Parameterised Function

- A function need not send any information to the called function.
- It is possible to write a python function without any parameters also.
- The following examples.

```
# Demonstration of a function  
# where no arguments are passed  
  
def warning_message():  
    print("The variable is not used")  
  
# function call  
warning_message()
```

Parameterised Functions

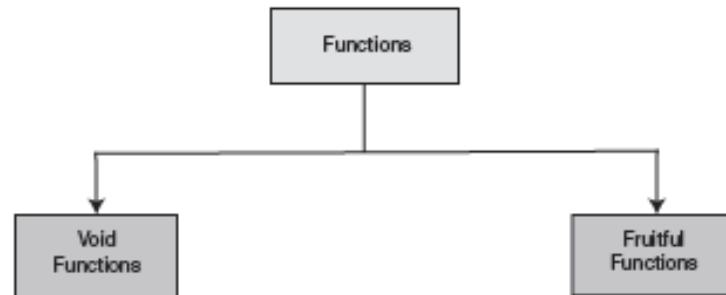
- A parameterised function passes information as arguments to the function.
- The following examples illustrates parameterised function where argument x is passed and its cube is obtained.

```
def cube(x):  
    print ("The cube is", x*x*x)  
  
cube(10)
```

```
C:\Users\harish\OneDrive\Desktop\Chapter 6>python listing3.py  
The cube is 1000
```

Void Functions and Fruitful (Value Returning) Functions

- There are two types of functions based on the return values: void functions and fruitful functions.



Write a python program to compute some arithmetic operations

```
def compute (a, b):  
    """ This function does basic computation """  
    print('{0} + {1} = {2}'.format(a,b,a + b))  
    print('{0} - {1} = {2}'.format(a,b,a - b))  
    print('{0} * {1} = {2}'.format(a,b,a * b))  
    print('{0} / {1} = {2}'.format(a,b,a / b))
```

```
compute (100, 200)
```

```
C:\Users\harish\OneDrive\Desktop\Chapter 6>python listing7.py  
100 + 200 = 300  
100 - 200 = -100  
100 * 200 = 20000  
100 / 200 = 0.5
```



Important Facts

It must be observed that the function call `compute (a, b)` makes the assignment `a=100` and `b=200`. Then, in the function, the computations are performed based on the value of `a` and `b`. If the order is reversed, say `(b, a)`, then the values would be different. So, the order of the parameter is important.

Fruitful Functions

- A function can return a value.
- A function that returns a value is called fruitful functions.
- Fruitful functions execute a set of statements given in the function body and return a value to the function call.
- A return function can be literal, variable, or expression. One can write a fruitful function for computing an area of a circle

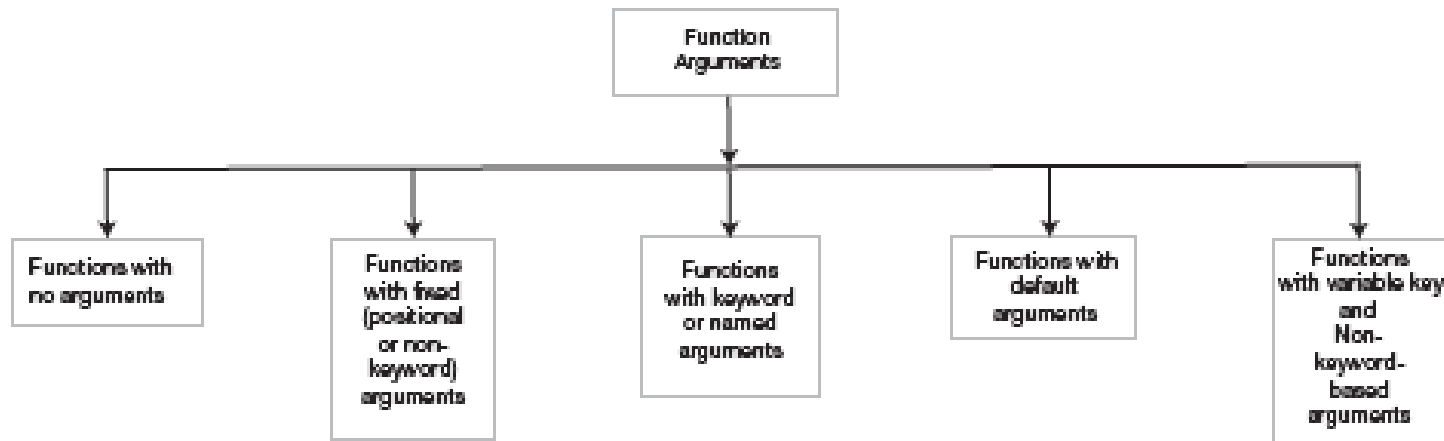
```
import math
def area (radius):
    return (math.pi*radius**2)

result = area(3)          # Return value is stored in a variable result
print("The area is", result)
```

```
C:\Users\harish\OneDrive\Desktop\Chapter 6>python listing8.py
The area is 28.274333882308138
```

Types of Function Arguments

- There are five types of functions based on function arguments,



Functions with No arguments

- This type of function has no arguments. There is no need to pass any arguments.
- Example:

```
def print_msg():  
    print("Observe no arguments are passed")  
  
print_msg()
```

```
C:\Users\harish\OneDrive\Desktop\Chapter 6>python listing11.py  
Observe no arguments are passed
```

Functions with Positional Arguments

- A functional call is required to invoke the functions.
- A Parameterised function can send a set of arguments to the function to invoke it.
- These arguments are known by various names such as
 1. Positional Arguments
 2. Required Arguments
 3. Non-keyword arguments
 4. Mandatory Arguments

These variables are stored in the parameters for further processing.

Keyword Arguments

- Keyword arguments are also known as named arguments.
- These are the parameters that are identified by name.
- One knows that the name is assigned by the assignment statement.
- Therefore, one uses an assignment statement for assigning values.

In the above program, the value can be provided using the assignment as `Compute (a=100,b=200)`

Here, a and b are keyword arguments. The assigned values are passed to the functions from the function call. The advantage of keyword arguments is that one has more control and the value is provided by named arguments.

For Example 11, the value can be provided as

```
compute(a=100, b=200)
```

The advantage is that even if the positional order is changed as

```
compute(b=100, a=200)
```

the results would be the same. In other words, the major disadvantage of positional arguments is negated by keyword arguments as there is no need to match the order of the parameters that are defined in the function header.

Default Arguments

- If the value of the arguments is not given at the time of call, the argument can be initialised with the default value. A default parameter is a keyword parameter that has a default value. This is a situation where the value is already known. For example, if the value of b and c are known already one can default value as compute(a, b=10, c=30) in functional parameters. So, there is no need to pass the value for these arguments. The following script illustrates this:

```
def compute(a, b=10, c=30):  
    sum = a+b+c  
    print (f"sum is {sum}")  
    return  
  
compute(10) # Observe values of b and c are not given
```

```
C:\Users\harish\OneDrive\Desktop\Chapter 6>python listing16.py  
sum is 50
```

Passing of object Reference

- In most Programming languages, there are two types of communication between the function caller and function definition.
- They are
 1. Pass by value(or call by value)
 2. Pass by reference (or call by reference)

Namespaces, global variables, and Scope

- The concept of namespace and scope help to identify variable hierarchy in the Python context. A namespace is a naming system and it is also known as context.
- A namespace is a directory, mapping names to values. In other words, a name is a key to the dictionary. It is mapped to values.
- When a new function is defined, a new namespace is created. All namespaces can coexist at the same time but are isolated from each other.

Some of the namespaces are as follows.

- 1) Global namespace: these are the names of the modules
- 2) Local namespace: the names in the function or function calls
- 3) Built-in namespace: all the names of the built-in functions and exceptions

Namespace Details

No	Namespace	Contents of Namespace	The life period of the namespace
1.	Global namespace	Names of the modules	From the module import till the Python interpreter quits
2.	Local namespace	All the variables that are associated with the function	From the function invocation to the function return or exception happens within the function
3	Built-in namespace	All the Python built-in objects	From the Python interpreter starts till it exits

Non-local variables

- Non-local variables are the newest in python.
- The scope of the nonlocal variables is not defined.
- This makes the variable neither global nor local.
- It is neither present in the local scope nor the global scope.
- The keyword creates the nonlocal variables.
- If the non local variables are modified, they are manifested in the local variables too.

Conflict Resolution in Scope using the LEGB Rule

- The types of scopes are given below.
 1. Built-in
 2. Global
 3. Enclosing
 4. Local