# Chapter 6 User-defined Function

A function is a set of statements that take inputs, do some specific computation, and produce output. The idea is to write commonly used code once and then reuse instead of writing the same code again and again for different inputs.

- Functions that readily come with Python such as `print`(), `input`(), `eval`(), etc., are called **built-in functions**.
- If we use functions written by others in the form of module/library such as `sqrt`() from math module, `randint`() from random module, etc., they are called **module function** or **library functions**.
- All the other functions that we write on our own fall under **user-defined functions**. Our user-defined function could be a module function or library function to someone else.

## 1. Defining a Function

A function definition consists of the function's name, parameters, and body.

The syntax for defining a function is as follows:

```
def function_name(parameter1, parameter2, ...):  # function header
   # function body
```

A function contains a header and body. The header begins with the **`def`** keyword, followed by the function's name and parameters, and ends with a colon. The variables in the function header are known as **formal parameters** or simply **parameters**. Parameters are optional; a function may not have any parameters.

The function body contains a collection of statements that define what the function does.

Some functions return a value, while other functions only perform desired operations without returning any value.

- A function that returns a value is called a **value-returning function**.
- A function that does not return a value is called a **void function** or **none function**.

**Example 01:** Create a value-returning function that sums two number together and return the result.

```
# Define function
def sum1(num1, num2):

    return num1 + num2
```

The **`return`** keyword is required for a value-returning function to return a result. The function terminates when a return statement is executed.

**Example 02:** Create a none function that sums two number together and display the result.

```
# Define function
def sum2(num1, num2):
    total = num1 + num2

    print("The total is", total)
```

## 2. Calling a Function

In a function's definition, you define what it is to do. To use a function, you have to call or invoke it. The program that calls a function is called a **caller**. When a function is invoked/called, you pass a value to the formal parameter. This value is referred to as an **actual parameter** or **argument**.

There are two ways to call a function, depending on whether or not it returns a value.

- If the function returns a value, a call to that function is treated as a value. For example,

  ```
  total = sum1(3, 4)
  ```

  or

  ```
  print(sum1(3, 4))
  ```

- If the function does not return any value, a call to that function is treated as a statement. For example,

  ```
  sum2(3, 4)
  ```

**Example 03:** Create a function that sums two number together and return the result.

```
# Define function
def sum1(num1, num2):

    return num1 + num2

# Call the function
total = sum1(5.5, 20)

# Display the result
print("The total is", total)
```

**Output**

```
The total is 25.5
```

**Example 04:** Create a function that sums two number together and display the result.

```
# Define function
def sum2(num1, num2):
    total = num1 + num2

    print("The total is", total)

# Call the function
sum2(5.5, 20)
```

**Output**

```
The total is 25.5
```

**Note**: A value-returning function also can be invoked as a statement. In this case, the return value is ignored. This is rare but is permissible if the caller is not interested in the return value. For example,

**Example 05:**

```
def sum(num1, num2):
    print("Function sum is called.")

    total = num1 + num2

    return total

sum(1, 2)
```

**Output**

```
Function sum is called.
```

**Note**: Technically, every function in Python returns a value whether you use return or not. If a function does not return a value, by default, it returns a special value **`None`**. For example,

**Example 06:**

```
def sum(num1, num2):
  total = num1 + num2

print(sum(1, 2))
```

**Output**

```
None
```

### 3. Returning Multiple Values

Python allows a function to return multiple values.

**Example 07:** defines a function that takes two numbers and returns them in ascending order.

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)

print("n1 is", n1)
print("n2 is", n2)
```

**Output:**

```
n1 is 2
n2 is 3
```

## 4. The Scope of Variables

The **scope** of a variable is the part of the program where the variable can be referenced.

A variable created inside a function is called a **local variable**. Local variables can only be accessed within a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.

In Python, you can also use **global variables**. They are created outside all functions and are accessible to all functions in their scope. Considering the three following examples:

```
var1 = 1  # global variable

def f1():
    var2 = 2  # local variable

    print(var1)  # displays 1
    print(var2)  # displays 2


f1()  # call function f1()

print(var1)  # displays 1
print(var2)  # Out of scope, so this gives an error
```

## 5. Default Parameters

Python allows you to define functions with default parameter values. The default values are passed to the parameters when a function is invoked without the parameters.

**Example 08:** demonstrates how to define functions with default parameter values and how to invoke such functions.

```
def print_area(width = 1, height = 2):
  area = width * height
  print("width:", width, "\theight:", height, "\tarea:", area)

print_area()
print_area(4, 2.5)
print_area(height = 5, width = 3)
print_area(width = 1.2)
print_area(1.5)
print_area(height = 6.2)
```

**Output:**

```
width: 1        height: 2       area: 2
width: 4        height: 2.5     area: 10.0
width: 3        height: 5       area: 15
width: 1.2      height: 2       area: 2.4
width: 1.5      height: 2       area: 3.0
width: 1        height: 6.2     area: 6.2
```

**Note**: A function may mix parameters with default parameters and non-default parameters. In this case, **the default parameters must be declared last**. For example:

```
def fun1(x = 1, y, z = 2):   # error
def fun2(x, y = 1, z):       # error
def fun3(x, y, z = 2):
```

## 6. Variable-Length Parameter

A function with variable length parameter **can accept zero or multiple arguments**. Variable length parameters are most useful when the number of arguments to be passed to the method is not known beforehand. They also reduce the code as overloaded methods are not required.

**Example 09:** Using Variable-Length Parameter

```
def sum(*numbers):
    total = 0
    for num in numbers:
        total += num
    return total

print(sum())
print(sum(1))
print(sum(1, 2, 3, 4, 5))
```

**Output**

```
0
1
15
```

**Note:** that **only one** variable-length parameter may be specified in a function, and this parameter **must be the last parameter**. Any regular parameters must precede it. For example,

```
func1(*scores, name)    # error

func2(name, *scores)
```

## 7. Immutable Objects vs. Mutable Objects

In Python, integers, floats, strings, lists, dictionaries, etc., are objects. When you pass an object to a function, the reference to the object is passed. However, there is an important difference between passing immutable objects and mutable objects.

- **Immutable objects** (e.g., integer, float, string): These cannot be modified. When you pass an immutable object to a function, any operation that appears to "modify" the object will actually create a new object instead, leaving the original object unchanged.
- **Mutable objects** (e.g., lists, dictionaries): These can be modified. When you pass a mutable object to a function, changes made to the object within the function will affect the original object because the function operates on the same reference.

**Example 10:** Pass immutable objects to a function

```
def update(x):
    x = 10;

# Test program
x = 1
update(x)

print("After calling the function")
print("x =", x)
```

**Output**

```
After calling the function
x = 1
```

| x |
|---|
| 1 |
| 0x6dfed4 |

| update::x |
|---|
| ~~1~~ 10 |
| ~~0x6dfed4~~ 0x9rg768 |

**Example 11:** Pass a mutable object to a function

```
def update(list1):
    list1[0] = 100

# Test program
list1 = [1, 3, 5]
update(list1)

print("After calling the function")
print(list1)
```

**Output**

```
After calling the function
[100, 3, 5]
```

| list1, update::list1 |
|---|
| ~~[1, 3, 5]~~ [100, 3, 5] |
| 0x6dfed4 |

## 8. Modules

Module is a file containing a set of functions. The module can be imported into a program for reuse.

Types of modules in Python:

1. **Library modules**
    - These are pre-defined modules in Python. They are further divided into:
        - **Standard library modules**: These come pre-installed with Python and do not require installation. Example: **random** and **math**.
        - **Third-party modules**: These require installation. Example: **numpy** and **panda**.

2. **User-defined modules** are modules defined by users.

### 8.1 Defining Modules

To create a module just save the code you want in a file with the file extension .py. For example, save this code in a file named mymodule.py. Let's place this module file in the same directory with your program that needs to use this module.

```
def greeting(name):
    print("Hello, " + name)

def square(a):
    return a * a
```

Note: Many programming languages allows you to define more than one functions with the same name in a module, but it is not supported in Python. If you define multiple functions with the same name in Python, the later definition will replace the previous ones.

### 8.2 Using Modules

Now we can use the module we just created, by using the `import` statement.

**Example 12:** import the module named mymodule, and call the greeting function:

```
import mymodule

mymodule.greeting("John")

print(mymodule.square(10))
```

**Output**

```
Hello, John
100
```

### 8.3 Re-naming a Module

You can create an alias when you import a module, by using the `as` keyword. For example,

**Example 13:** Create an alias for `mymodule` called `mx`:

```
import mymodule as mx

mx.greeting("John")
```

**Output**

```
Hello, John
```

### 8.4 Importing Module

You can choose to import only parts from a module, by using the `from` keyword.

**Example 14:** the module named `mymodule` has more than one functions, and you would like to import only the function from the module.

```
from mymodule import greeting

greeting("John")
```

**Output**

```
Hello, John
```

Note: When importing using the `from` keyword, do not use the module name when referring to elements in the module. Example: `greeting("John")`, not `mymodule.greeting("John")`

**Example 15:** The module named `mymodule` has more than one functions, and you would like to import only the function from the module.

```
from mymodule import greeting, square

greeting("John")

print(square(10))
```

**Output**

```
Hello, John
100
```

**Example 16:** Import everything from `mymodule`.

```
from mymodule import *

greeting("John")

print(square(10))
```

**Output**

```
Hello, John
100
```

**Exercises**

1. Write a function called `display_sorted_numbers`(num1, num2, num3) that displays three numbers in increasing order. Then write a test program that asks the user to enter three numbers and invokes the function to display them in increasing order. Here is a sample run:

   ```
   Enter three numbers: 3, 2.4, 5
   The sorted numbers are:  2.4  3  5
   ```

2. Write a function called `display_longest_word` that will accept a string, then displays the longest word within the string.
3. Write a function called `remove_items` that will accept an item and a list, remove all the occurrences of the item from the list, then returns the list.
4. Write a function called `remove_dupplicates` that will remove the duplicates for an integer list that is passed to it.
5. (Twin primes) Twin primes are a pair of prime numbers that differ by 2. For example, 3 and 5 are twin primes, 5 and 7 are twin primes, and 11 and 13 are twin primes.

   Create a function called `is_prime` that checks whether a given number is a prime or not. Then, create another function called `generate_twin_primes` that returns all the twin primes less than 1200. Write a test program that displays all twin primes less than 1200. Here is a sample run:

   ```
   The twin prime numbers which are less than 1200:
   (3, 5)
   (5, 7)
   ...
   ```

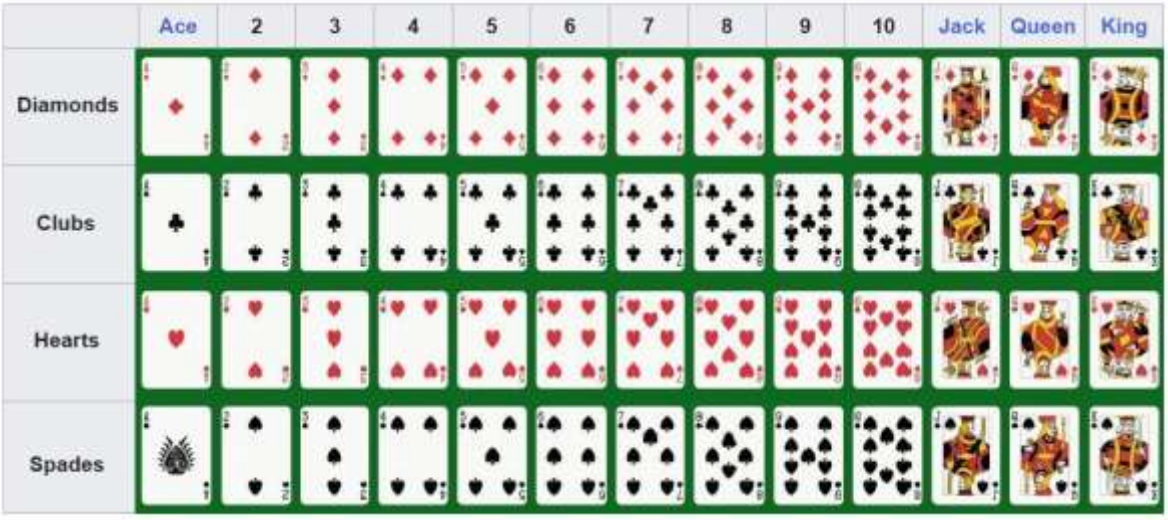
6. A palindrome is a word, number, or other sequence of characters which reads the same backward as forward. Write a method called `generate_palindromes` that displays all four-digits palindrome numbers.
7. (Anagram) Two words are anagrams if they contain the same letters in different orders, for example, binary and brainy. Write a method called `is_anagram` that takes two strings and returns `True` if they are anagrams, otherwise, returns `False`.
8. Write a method called `second_largest` that takes in a 2D list of integer numbers and find the value of the second largest number stored in the list.
9. Write a method called `display_leaders` that accepts a list of integers, then display all the LEADERS in the list. An item is a leader if it is greater than all the items to its right side. And the rightmost item is always a leader. Write a test program that creates a list of 10 integers, and call the function to displays the leaders.

10. Write a module that contains the following two functions:
    - `compute_circle` that calculates the area and perimeter of a circle, and returns the results.
    - `compute_traingle` that calculates the area and perimeter of a triangle, and returns the results.

    Write a test program that will ask the user to select the menu:
    1. Circle
    2. Triangle

    After the user select the menu, ask the user to enter what needed and display the result returned by the function in the module.

11. Write a program to simulate two players (user and computer) playing a card game (ដុកខាំង / ប៉ក់). Here is a standard 52-cards deck:




- Create the above 52 cards and store them in a list.
- When the game starts, the deck is shuffled.
- Dealing rules:
    - Dealing from the top (first item) of the deck. Dealing starts from the user.
    - When dealing cards, the cards are removed from the deck.
- Counting points rules:
    - Any picture card (jack, queen, or king) is equal to 0 point.
    - Points are counted by summing card points and taking the remainder when divided by 10. For example, 10 points = 0 point, 15 points = 5 points, 26 points = 6 points.
- Boom rules:
    - If the user or computer, or both have 8 or 9 points from the initial two cards, it is a "Boom!!!". No third card is drawn, and the result (It is a draw, User wins, or User loses) is immediately declared.
- Third card rules:
    - Third cards are drawn from the bottom (last item) of the deck.
    - The user decides whether to draw a third card first.

- The computer then decides to draw a third card based on its current points:
  - Points < 4: it **must** draw a third card.
  - Points = 4: it has an 80% chance of drawing and a 20% chance of skipping.
  - Points = 5: it has a 40% chance of drawing and a 60% chance of skipping.
  - Points = 6: it has a 10% chance of drawing and a 90% chance of skipping.
  - Points ≥ 7: it will **not** draw a third card.

  Hint: use `randint()` function defined in `random` module to generate a random number between 0 and 99, and check if it is less than a specific percent.

```
import random

number = random.randint(0, 99)

if number < 80:  #   80% chance
     # draw a third card
```

Note that the computer's cards are revealed only when the game ends.