Chapter 3: Functions and Modules

Chapter Objectives

By the end of this chapter, the student will be able to:

- Define a function
- Use parameters and return a value
- Understand local and global variables
- Use positional, named, and default arguments
- Import a standard module
- Create a custom module

3.1 Functions

1. Function Definition

A function is a reusable block of instructions that starts with the **def** keyword, specifying the function's parameters in parentheses, and the function header ends with a colon (:).

General Syntax

```
def function_name(parameters):
    instructions
    return value
```

 # Function call

function_name(arguments)

- **def**: Required keyword
- **parameters**: Arguments received
- **return**: Returns a result (optional)

Note: The function definition must precede its use.

a) Function Parameters

Several types of parameters can be distinguished:

Example 1: A function without parameters that prints a "Hello world" message.

```
def message():
    print("Hello world")

message()  # Function call, output: Hello world
```

Example 2: A function with two parameters a and b.

```
def somme(a, b):
    s = a + b
    return s

som = somme(2, 4)  # Function call
print(2, '+', 4, '=', som)

# Output: 6
```

Note: To call a function, use the function name followed by parentheses.

- The **return** statement returns the function's value, ends its execution, must be inside the instruction block, and is not mandatory.

Example: Write a function to display whether a number is even or odd.

```
# Function definition
def even_odd(n):
    if n % 2 == 0:
        response = 'even number'
    else:
        response = 'odd number'
    return(response)

# Function call
a = 17
print(a, 'is an', even_odd(a))
# Result: 17 is an odd number
```

b) Function Arguments

1. Positional Arguments

The number and order of arguments in a function call are very important.

Example 1

```
def subtraction(a, b):
    print(a - b)

# Function call
subtraction(10, 5)  # Output: 5
```

Example 2

```
def soustraction(a, b):
    print(a - b)

# Function call
soustraction(5, 10)  # Output: -5
```

2. Keyword Arguments (Labeled)

You can call a function with some or all arguments out of order as long as you specify their names:

Example 3

```
def calcul(a, b, c):
    return (a + b * c)

# Function calls
print(calcul(1, 2, 3))            # Output: 7
print(calcul(b=1, c=2, a=3))  # Output: 6
print(calcul(1, c=2, b=3))      # Output: 7
```

Note:

With keyword arguments, the order does not matter, but the number of arguments must be respected.

When using both keyword and positional arguments simultaneously, all positional arguments must come before keyword arguments.

2. Recursive Function

A recursive function is one that calls itself multiple times.

Example: Factorial of n

```
def factorielle(n):
    if n = = 1:
        return 1
    else:
        return n * factorielle(n - 1)

print(factorielle(4))  # Function call
# Output: 24
```

PART II: MODULES IN PYTHON

1. Introduction to Modules

A module is a Python file (.py) containing definitions and statements such as :

- Functions
- Variables
- Classes.

Modules allow code reuse and help structure large programs into smaller, manageable components.

- Advantages of using modules:
- Code reusability
- Better organization
- Improved readability
- Simplified maintenance

2. Importing Standard Modules

2.1 Importing the Entire Module

Syntax:

```
import module_name
```

Example:

```
import math
print(math.sqrt(25))
print(math.pi)
```

output : 5.0
3.141592653589793

2.2 Importing with Alias

Syntax:

```
import module_name as alias
```

Example:

```
import math as m
print(m.sqrt(25))
```

2.3 Importing Specific Functions

Syntax:

```
from module_name import function_name
```

Example:

```
from random import randint
print(randint(1,6))
```

2.4 Importing Multiple Functions

```
from math import sqrt, pi, sin
```

2.5 Avoid Using * Import

Using 'from module import *' is not recommended because it can create name conflicts and reduce code readability.

3. Common Built-in Modules

Module	Utility
math	Mathematical operations (sqrt, pi, sin, cos, etc.)
random	Random number generation (randint, random, choice)
statistics	Statistics (Mean, median, variance)
os	Interaction with operating system
datetime	Date and time manipulation

4. Using standard modules

a) Math module

```
import math
r = 3
area = math.pi * r**2
print(area)
```

b) Random module

Simulation of a dice roll (de dé):

```
from random import randint
print(randint(1,6))
```

c) Statistics module

```
import statistics
grades = [10, 12, 15, 8, 14]
print(statistics.mean(grades))
```

5- Package

A package is a folder containing several modules.

Example:

```
import matplotlib.pyplot as plt
```

• matplotlib = package

• pyplot = module

6. Creating a Custom Module

Step 1: Create a file named 'mymodule.py'

Example content of mymodule.py:

```
def  add(a, b):
    return a + b
```

```
def  multiply(a, b):
    return a * b

def  square(x):
    return x * x

if __name__ == '__main__':
    print(' Test the module when run directly')
```

Step 2: In another **main.py** file (same folder), import and use the functions.

```
import mymodule
print(mymodule.add(3,4))    # output 7
print(mymodule. multiply (3,4))    # output  12
print(mymodule.square(5))   # output 25
```

Run **python main.py**

5. The Main Block Structure

When writing modules, it is good practice to include the following structure:

```
if __name__ == '__main__':
    print(' Test the module when run directly')
```

This ensures that the test code runs only when the file is executed directly.

8. Chapter Summary

Modules are essential tools for structuring Python programs. They promote reusability, readability, and maintainability. Python provides many built-in modules, and programmers can also create their own.

6. Mini Project

- Create a module called 'geometry.py' containing the following functions:
- area_rectangle(length, width)
- perimeter_rectangle(length, width)
- area_circle(radius)

Then create a main program that imports the module and tests all functions.

Créer le module

Créez un fichier math_utils.py contenant troix fonctions basiques : add(), multiply() et is_even().

1-"""Simple math utilities module."""

```
def add(a, b):
    """Return the sum of two numbers."""
    return a + b

def multiply(a, b):
    """Return the product of two numbers."""
    return a * b

def is_even(n):
    """Check if a number is even."""
    return n % 2 == 0

 if __name__ == "__main__":
    # Test the module when run directly
    print("2 + 3 =", add(2, 3))
    print("4 * 5 =", multiply(4, 5))
    print("6 is even:", is_even(6))
```

2- This file becomes a reusable module.

Import the module

In another main.py file (same folder), import and use the functions.

```
import math_utils
result1 = math_utils.add(10, 20)
result2 = math_utils.multiply(7, 8)
```

```
print(f"10 + 20 = {result1}")  # Output: 30
print(f"7 * 8 = {result2}")    # Output: 56
print(f"10 is even: {math_utils.is_even(10)}")  # Output: True
```

Run **python main.py**

7. Exercises with Solutions

Exercise 1: Calculate the square root of 49 using math module.

Solution:

```
import math
print(math.sqrt(49))
```

Exercise 2: Generate a random number between 10 and 20.

Solution:

```
from random import randint
print(randint(10,20))
```

Exercise 3: Compute the mean of a list using statistics module.

Solution:

```
import statistics
print(statistics.mean([10,12,14]))
```