Chapter III

Functions and Modules

1. Function Definition: Syntax, Parameters, Return Values

In mathematics, a ***function*** is a rule that takes an input and produces an output.

In Python, a function works in a very similar way: it allows us to group a set of instructions under a single name so that we can reuse them whenever needed.

So, a function helps us:

- Organize our code clearly.
- Avoid repetition.
- Break a complex problem into smaller, manageable parts.

Basic syntax:

In Python, a function is defined using the keyword "**def**":

```
def function_name(parameters):
    # Instructions
    return result
```

- ✓ ***function_name*** is the name we give to the function.
- ✓ ***parameters*** are inputs to the function (like variables in mathematics).
- ✓ The ***return*** statement sends a result back to the place where the function was called.

Example 1: ***Mathematical function***

If we define the mathematical function: $f(x) = 4x^2 + 3x + 7$

In Python, we can write:

```
def f(x):
    result = 4*x**2+3*x+7
    return result
```

Example 2: ***Circle Area***

If we define the mathematical function: $f(radius) = 3{,}14(radius)^2$

In Python, we can write:

```
def f(radius):
    result = 3,14159*radius**2
    return result
```

2. Using Functions: Function Calls and Argument Management

One a function is defined; we can use it by calling it.

2.1 Function Call:

Calling a function means giving it a value (or values) so it can perform its task.

```
result = f(3)
print(result)
```

In this case, "3" is called an ***argument*** (The actual value given to the function).

2.2 Default Arguments:

Sometimes, we want a function to have a default value if the user does not provide one.

```
def power(x, n=2)
   result=x**n
   return(result)
```

if we call:

```
power(5)
```

The result will be 5^2, because ***n*** defaults to ***2***.

2.3 Named Arguments:

Python also allows us to specify arguments by name (Order doesn't matter):

```
def power(x, n=2)
   result=x**n
   return(result)
```

if we call:

```
power(n=3, x=4)
```

The result will be 64.

3. Module Import and Creation: Standard Modules, Custom Modules

As programs grow larger, it becomes important to organize code into separate files. This is where modules become useful.

A module is simply a Python file that contains functions, variables, or other definitions that can be reused in another program.

3.1 Importing Standard Modules:

Python provides many built-in modules.

For example, the ***math*** module gives access to mathematical functions:

```
import math
print(math.sqrt(16))
print(math.pi)
```

this allows us to use advanced mathematical tools without writing them ourselves.

We can also import specific functions:

```
from math import sqrt
print(sqrt(25))
```

3.2 Creating a Custom Module:

We can also create our own module.

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

```
def square (x):
    return x**2
def cube (x):
    return x**3
```

Step 2: Use it in another file:

```
import mymodule
print(mymodule.square(4))
```

This is similar to defining a set of mathematical tools in one notebook and reusing them later.