

Basic Programming with Python

Prepared By:

Professor Dr. Md. Mijanur Rahman
Department of Computer Science & Engineering
Jatiya Kabi Kazi Nazrul Islam University
www.mijanrahman.com

2

Introduction To Python Programming

CONTENTS

2.1. Python Programming	1
2.1.1 Major Features of Python.....	2
2.2. Basic Structure of Python Program	2
2.2.1 Components of Python Program.....	4

2.1. PYTHON PROGRAMMING

Python is a high-level, interpreted, dynamic, and object-oriented programming language. It is a general-purpose language that is widely used for a wide range of tasks, including web development, data analysis, machine learning, scientific computing, and much more. The language was first released in 1991 by Guido van Rossum, and its design philosophy emphasizes code readability and simplicity.

One of the key features of Python is its easy-to-learn syntax and dynamically-typed nature, which allows developers to write code quickly and focus on the problem they are trying to solve rather than getting bogged down by the syntax. Additionally, Python has a vast and growing library of modules and packages that make it easy to perform tasks such as connecting to web servers, reading and writing data from databases, and performing complex mathematical calculations.

Python is also widely used for artificial intelligence and machine learning applications, and its simplicity and ease of use have made it one of the most popular programming languages for data scientists and machine learning engineers. The language provides a variety of libraries and tools for tasks such as data visualization, numerical computing, and machine learning, making it a great choice for those looking to build AI applications.

2.1.1 Major Features of Python

The key features of the Python programming language include:

1. **Easy-to-learn syntax:** Python has a simple and easy-to-learn syntax, making it a great choice for beginners. The language's design philosophy emphasizes code readability, which makes it easier to understand and maintain code over time.
2. **Dynamic typing:** Python is a dynamically-typed language, which means that the data type of a variable can change during the course of the program. This feature allows for greater flexibility and ease of use, as developers don't have to worry about explicitly defining the data type of a variable before using it.
3. **Large standard library:** Python has a large and growing standard library, which includes modules and packages for a wide range of tasks, such as connecting to web servers, reading and writing data from databases, and performing complex mathematical calculations.
4. **Object-oriented programming:** Python is an object-oriented language, which means that it supports the creation and manipulation of objects. This feature makes it easier to write organized and reusable code and to structure programs in a way that makes sense for the problem being solved.
5. **Interpreted language:** Python is an interpreted language, which means that code is executed line-by-line rather than being compiled into machine code before being executed. This feature makes it easier to debug code, as errors can be detected and corrected more quickly.
6. **Great for data analysis and machine learning:** Python has become one of the most popular programming languages for data analysis and machine learning thanks to its simplicity, ease of use, and powerful libraries and tools for these tasks.
7. **Cross-platform compatibility:** Python is a cross-platform language, which means that code written on one platform can run on any other platform with a Python interpreter installed. This feature makes it easier to share code and collaborate with others.

2.2. BASIC STRUCTURE OF PYTHON PROGRAM

Python provides a very basic and simple structure for writing a program. It consists of different sections. Some sections are compulsory, and some are optional. We can include or exclude the optional sections as per requirement or as per the situation. Figure 2.1 shows the basic structure of the Python program, and a brief explanation of each section in the Python program structure is as follows:

1. **Documentation section:** The documentation section includes comments that specify the aim of the program. We write comments in a program to improve the readability of the program.
2. **Import statements:** This section includes various in-built or user-defined modules in different modules so that we can use functionality already defined in the existing module.
3. **Global declaration section:** In this section, we define global variables for the programs. A global variable is a variable that we can access from anywhere in the program.

4. **Class section:** This section tells the information about the user-defined classes present in the program. A class is a group of variables (called data members) and member functions (called methods) that work on data members. It contains the class definition, data members, and methods definition.
5. **Subprogram section:** In this section, we define user-defined functions. A function is a set of statements that will execute when we call it from anywhere in the program.
6. **Playground section:** This is the main section where we call user-defined functions and class methods. Moreover, we create an object of the class in this section.
In Python, there is no main function (i.e., main method) that separates the other section from the playground (main) section. We can separate it by writing a comment line between the playground and other sections for a better understanding.

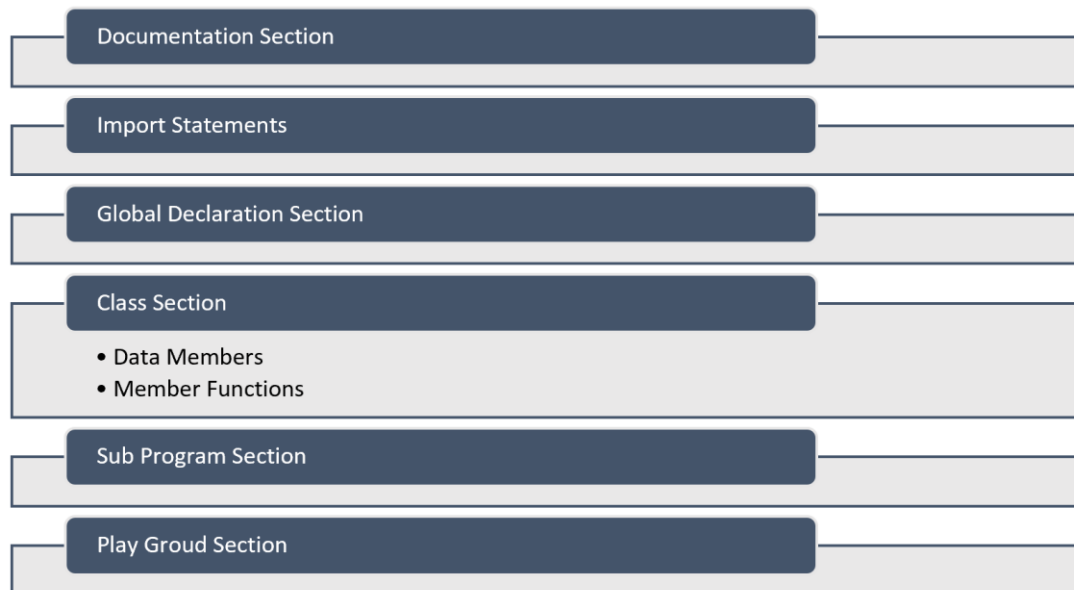


Figure 2.1: A simple structure of a Python program.

Example 2.1: Hello World Program in Python

The following is a simple "Hello World" program in Python:

```
# This is a 'Hello World' program in Python.  
print("Hello, World!")
```

In this program, the first line specifies the comment telling the purpose of the program. It is not an executable statement. The second line uses a `print()` function to display the string "Hello, World!". When you run this program, it will display the message "Hello, World!" in the output.

Output:

```
Terminal  
Hello, World!
```

This is a basic introductory program that is often used to demonstrate the syntax and structure of a programming language.

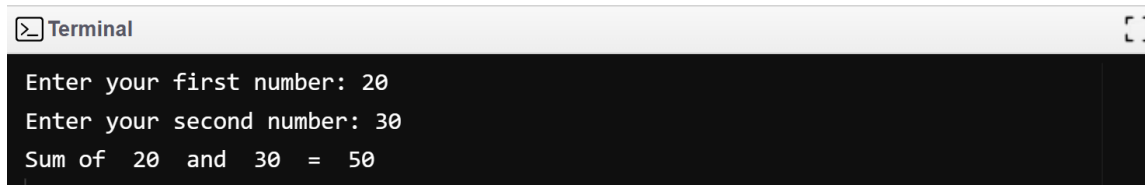
Example 2.2: Compute Sum of Numbers in Python

The following is an example program that will find the sum of two numbers in Python.

```
# Python program to calculate the sum of two numbers.
x = int(input("Enter your first number: "))
y = int(input("Enter your second number: "))
sum = x + y
print("Sum of ", x, " and ", y, " = ", sum)
```

In this example, the first line specifies the comment telling the purpose of the program. It is not an executable statement. The second and third lines create variables `x` and `y`, and stores the value of two number entered by the programmer. The fourth line calculates the sum of the first number and the second number and stores the outcome in the third variable 'sum'. The fifth line displays the outcome.

Output:



```
Terminal
Enter your first number: 20
Enter your second number: 30
Sum of 20 and 30 = 50
```

2.2.1 Components of Python Program

The basic structure of a Python program includes the following components:

1. **Shebang line:** It is an optional line that specifies the location of the Python interpreter on the system. It starts with `#!` followed by the path to the Python interpreter. Example:

```
#!/usr/bin/env python
```

In this case, the shebang line indicates that the script should be run using the 'python' interpreter. The '/usr/bin/env' part is a common way to locate the interpreter in a portable manner, as it relies on the environment's PATH variable to find the appropriate interpreter.

2. **Comments:** In Python, we can add comments to the code to provide explanations, and documentation, or to disable certain lines of code. Comments are ignored by the Python interpreter and are meant for human readers.

The following are the two ways to write comments in Python:

- a. **Single-line comments:** To add a comment that spans only a single line, we can use the `#` symbol. Everything following the `#` symbol on that line will be considered a comment. Example:

```
# This is a single-line comment in Python
```

- b. **Multi-line comments:** If we want to add a comment that spans multiple lines, we can enclose the comment in triple quotes (`"""` or `'''`). This is often referred to as a docstring and can be used as a multi-line comment. Example:

```
"""
This is a multi-line comment
"""
```

```
in Python using triple quotes.  
It can span multiple lines.  
"""
```

3. **Import statements:** These statements allow to import modules or libraries into the program. The imported modules can provide additional functionality and classes in the program. There are a few different ways to use import statements in Python:

a. Importing an entire module:

```
import module_name
```

This form of import statement allows to import an entire module. To access functions, classes, or variables from the module, we need to prefix them with the module name followed by a dot. Example:

```
import math  
result = math.sqrt(16)
```

b. Importing specific names from a module:

```
from module_name import name1, name2
```

This form allows to import specific functions, classes, or variables directly into the code, without needing to use the module name as a prefix. Example:

```
from math import sqrt, pi  
result = sqrt(16)
```

c. Importing an entire module with a different name:

```
import module_name as alias_name
```

This form allows to import a module and assign it a different name (alias), which can be useful if the original module name is long or conflicts with other names in the code. Example:

```
import math as m  
result = m.sqrt(16)
```

d. Importing all names from a module:

```
from module_name import *
```

This form imports all names (functions, classes, variables) from a module directly into the code. However, it is generally considered good practice to avoid using this form, as it can lead to name clashes and make the code less readable. Example:

```
from math import *  
result = sqrt(16)
```

4. **Function definitions:** Functions are blocks of code that perform specific tasks. They can be called multiple times from different parts of the program. Functions are defined using the "def" keyword followed by the function name, parameters, and a colon. Example:

```
def function_name(parameters):  
    # Function body  
    # Code block with instructions  
    # Optional return statement
```

2. Introduction to Python Programming

return value

5. **Variables:** Variables are used to store data in a program. Variables in Python can be dynamically typed, meaning that the type of data stored in a variable can change during the course of a program. We can assign a value to a variable using the assignment operator (=). Example:

```
length = 5
width = 3
area = length * width # calculation using variables
name = "Rahman"
message = "Hello, " + name # concatenating strings

print("The area is:", area)
print(message)
```

6. **Main code:** This is the main body of the program where we can write the code logic. This section can include conditional statements, loops, function calls, and other constructs. It typically contains the main logic and flow of the program. The main code is not inside any function or class definition.

In Python, the main code is often placed inside an `'if __name__ == "__main__":'` block. This is an example structure of a Python script with the main code:

```
# Import statements (if required)
import module1
import module2

# Function definitions (if required)
def some_function():
    # Function body
    pass

# Main code block
if __name__ == "__main__":
    # Code here will be executed when the script is run directly
    # It can include function calls, variable assignments, control structures, etc.

    # Example:
    print("This is the main code.")
    result = some_function()
    print("Result:", result)
```

7. **Output statements:** These statements are used to print output to the screen or to another output device. The "print" function is used to print output in Python. Example:

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

The structure of a Python program is flexible and there is no strict rule on the order in which these components should be placed, but it is a good practice to follow a standard structure to make your code more readable and maintainable.

Example 2.3: Demonstrating different types of comments in Python

```
# This is a single-line comment

"""
This is a multi-line comment.
It spans multiple lines.
"""

'''
This is also a multi-line comment.
It can be enclosed within single quotes.
'''

# Single-line comment at the end of a line
x = 5 # Assigning value 5 to variable x
print("The value of x:", x)

# Multi-line comment followed by code
'''
The following code calculates the sum of two numbers.
'''
num1 = 10
num2 = 20
sum = num1 + num2
print("Number 1:", num1)
print("Number 2:", num2)
print("The sum is:", sum)

# Comments for documentation
def add(x, y):
    """
    This function adds two numbers.

    Parameters:
    x (int): The first number.
    y (int): The second number.

    Returns:
    int: The sum of x and y.
    """
    return x + y

# Commented-out code
# print(add(5, 10))
```

This script includes single-line comments, multi-line comments, comments at the end of lines, comments for documentation (docstrings), and commented-out code.

```
Output Clear  
The value of x: 5  
Number 1: 10  
Number 2: 20  
The sum is: 30  
  
=== Code Execution Successful ===
```

Example 2.4: Demonstrating import statements in Python

```
# Importing an entire module  
import math  
  
x = 25  
y = math.sqrt(x) # Calculate square root  
print("The value of x:", x)  
print("Square root of x:", y)  
  
# Importing specific functions from a module  
from datetime import datetime  
  
current_time = datetime.now()  
print("Current time:", current_time)  
  
# Importing a module with an alias  
import random as rnd  
r = rnd.randint(1, 100) #Random integer between 1 and 100  
print("Random number:", r)  
  
# Importing specific functions from a module with an alias  
from time import time as current_time  
  
print("Current timestamp:", current_time())  
  
# Importing all functions from a module  
from statistics import *  
  
data = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  
print("Dataset:", data)  
print("Mean:", mean(data)) # Mean value  
print("Median:", median(data)) # Median value
```


Output

Clear

```
The value of x: 25
Square root of x: 5.0
Current time: 2024-03-21 17:49:38.651378
Random number: 29
Current timestamp: 1711043378.6549747
Dataset: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Mean: 55
Median: 55.0

=== Code Execution Successful ===
```

Example 2.5: Demonstrating variable assignment in Python

```
# Integer assignment
x = 5
print("Integer assignment:")
print("x is:", x) # Output: 5

# Float assignment
y = 3.14
print("\nFloat assignment:")
print("y is:", y) # Output: 3.14

# String assignment
name = "Alice"
print("\nString assignment:")
print("name is:", name) # Output: Alice

# Boolean assignment
is_valid = True
print("\nBoolean assignment:")
print("is_valid is:", is_valid) # Output: True

# Multiple assignments
a, b, c = 1, 2, 3
print("\nMultiple assignments:")
print("a is:", a) # Output: 1
print("b is:", b) # Output: 2
print("c is:", c) # Output: 3

# Swap values
x, y = y, x
print("\nSwapping values:")
print("x is:", x) # Output: 3.14
```

2. Introduction to Python Programming

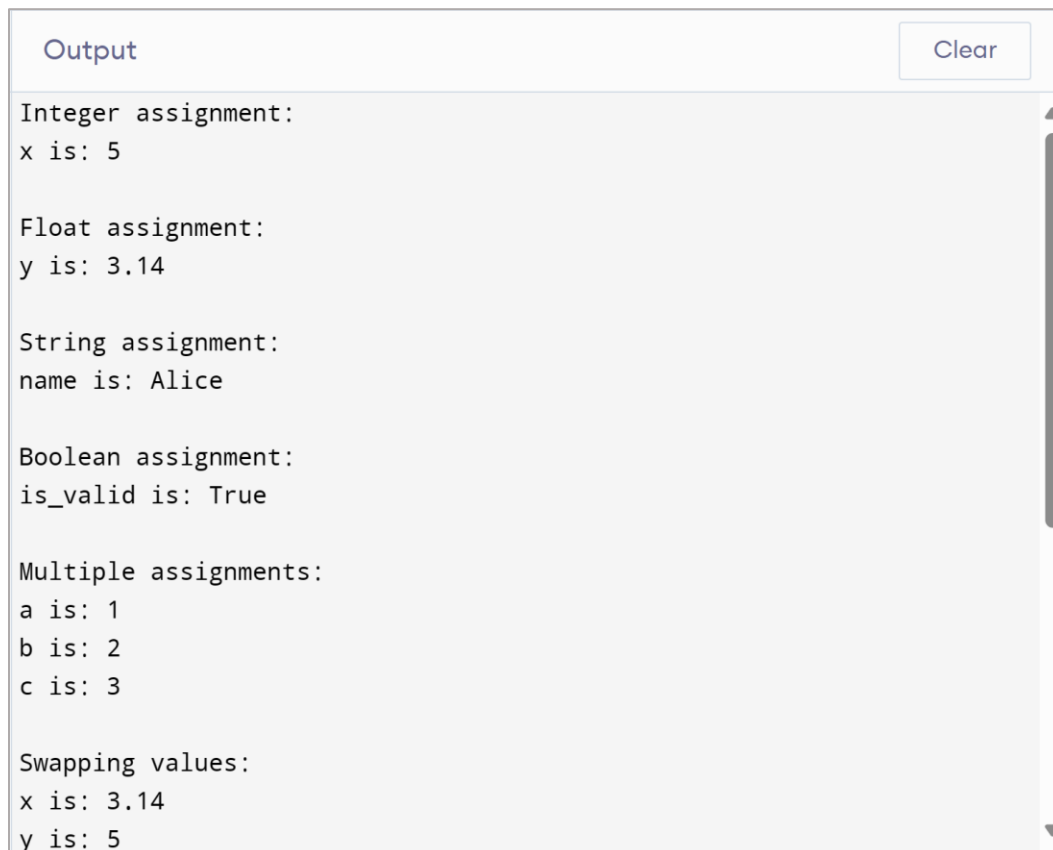
```
print("y is:", y) # Output: 5

# Assigning multiple variables the same value
print("\nAssigning multiple variables the same value:")
x = y = z = 10
print("x is:", x) # Output: 10
print("y is:", y) # Output: 10
print("z is:", z) # Output: 10

# Assigning a value to a variable using another variable
a = 5
b = a
print("\nAssigning a value to a variable using another variable:")
print("a is:", a) # Output: 5
print("b is:", b) # Output: 5

# Variable assignment in expressions
c = a + b
print("\nVariable assignment in expressions:")
print("c is:", c) # Output: 10
```

This script demonstrates various types of variable assignments including single assignments, multiple assignments, swapping values, assigning the same value to multiple variables, assigning values using other variables, and variable assignment in expressions.



The screenshot shows a window titled "Output" with a "Clear" button in the top right corner. The output text is as follows:

```
Integer assignment:
x is: 5

Float assignment:
y is: 3.14

String assignment:
name is: Alice

Boolean assignment:
is_valid is: True

Multiple assignments:
a is: 1
b is: 2
c is: 3

Swapping values:
x is: 3.14
y is: 5
```

Example 2.6: Demonstrating a function definition in Python

```
# Function definition
def greet(name):
    """
    This function greets the user with the given name.

    Parameters:
    name (str): The name of the user.

    Returns:
    str: A greeting message.
    """
    return f"Hello, {name}! Welcome to our program."

# Function call
user_name = input("Please enter your name: ")
greeting_message = greet(user_name)
print(greeting_message)
```

In this script, we define a function called `greet` that takes one parameter `name`. Inside the function, we use an f-string to generate a greeting message with the provided name. The function returns the greeting message. We then prompt the user to input their name. We call the `greet` function with the provided user name and store the returned greeting message. Finally, we print the greeting message to the console.

OutputClear

```
Please enter your name: Rahman
Hello, Rahman! Welcome to our program.

=== Code Execution Successful ===
```

Example 2.7: Demonstrating the Main Code Structure in Python

The following is a sample program in Python that demonstrates the main code structure:

```
# Import statements (if required)
import math

# Function definitions (if required)
def calculate_circle_area(radius):
    area = math.pi * radius ** 2
    return area

# Main code block
if __name__ == "__main__":
    # Code here will be executed when the script is run directly
```

2. Introduction to Python Programming

```
# Prompt the user for input
radius = float(input("Enter the radius of the circle: "))

# Calculate the area of the circle using the function
area = calculate_circle_area(radius)

# Print the result
print("The area of the circle is:", area)
```

In this sample program, the 'main code' prompts the user for the radius of a circle, calculates the area of the circle using the 'calculate_circle_area()' function, and then prints the result.

Output

Clear

```
Enter the radius of the circle: 5
The area of the circle is: 78.53981633974483

=== Code Execution Successful ===|
```