Programming Fundamentals [1]

A Study Guide for Students of Sorsogon State University - Bulan Campus[2]

Jarrian Vince G. Gojar[3]

January 22, 2025

[1] A course in the Bachelor of Technical Vocational Teacher Education (BTVTED) Major in Computer System Servicing.

[2] This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Programming Fundamentals.

[3] https://github.com/godkingjay

Sorsogon State University - Bulan Campus

Contents

List of Figures

List of Tables

List of Codes

Preface

> *"Programming is the art of telling another human being what one wants the computer to do."*
>
> – Donald Ervin Knuth

Jarrian Vince G. Gojar
https://github.com/godkingjay

1

Introduction to Python

1.1 Introduction

Python is a high-level, interpreted, interactive, and object-oriented scripting language. It is designed to be highly readable, using English keywords frequently, unlike other languages that use punctuation, and it has fewer syntactical constructions.

Python is a versatile, general-purpose, and powerful programming language. It is an excellent first language because it is concise and easy to read. Whether you want to do web development, machine learning, or data science, Python is the language for you.

1.1.1 History of Python

Python was developed by **Guido van Rossum** in the late 80's and early 90's at the National Research Institute for Mathematics and Computer Science in the Netherlands. Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

1.1.2 What is Programming Language?

A programming language is a formal language comprising a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms.

1.2 Environment Setup

In software development, an **integrated development environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Other than the IDE, it is also important to have the programming language installed on your computer.

1.2.1 How to Download and Install Python on Windows

To download and install Python on Windows, follow these steps:

1. Open a web browser and go to the official Python website at https://www.python.org/downloads/.

2. Click on the latest version of Python to download the installer.
3. Run the installer and follow the installation wizard.
4. Make sure to check the box that says “Add Python to PATH”.
5. Click on the “Install Now” button to install Python on your computer.

Figure 1: Python Installation Wizard

6. Once the installation is complete, you can verify the installation by opening a command prompt and typing `python -version`.

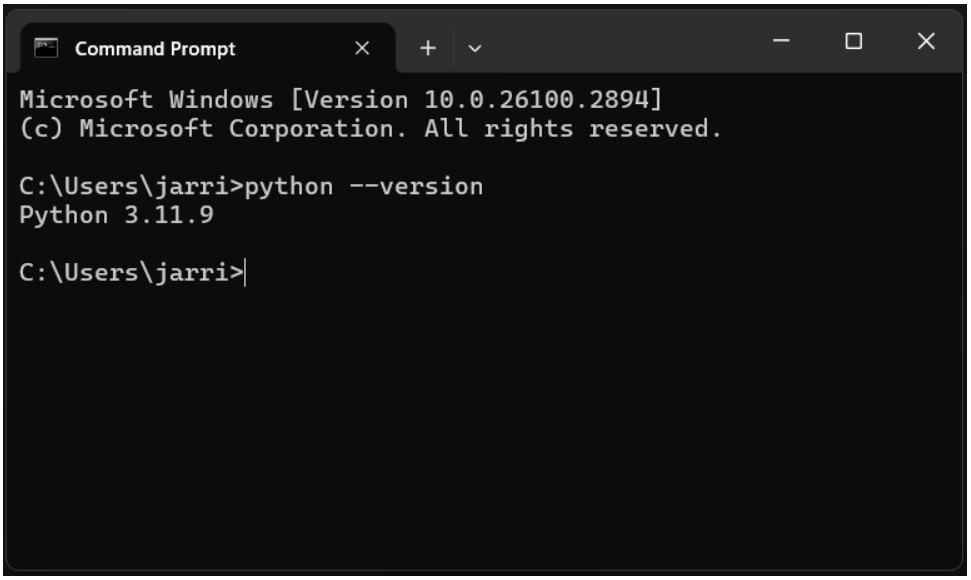

Figure 2: Python Installation Verification

1.2.2 How to Download PyCharm IDE on Windows

To download and install PyCharm IDE on Windows, follow these steps:

1. Open a web browser and go to the official PyCharm website at https://www.jetbrains.com/pycharm/download/.

2. Click on the "Download" button to download the installer.

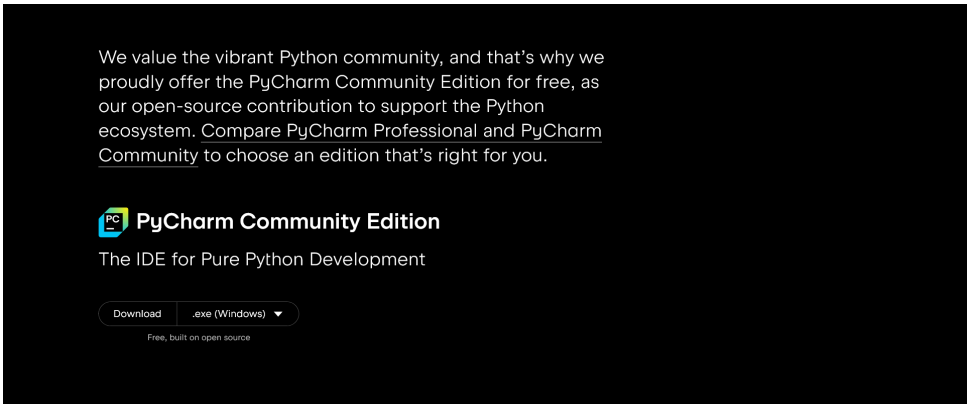

Figure 3: PyCharm Community Edition Download

3. Run the installer and follow the installation wizard.

4. Make sure to check the box that says "Create associations".

5. Click on the "Install" button to install PyCharm on your computer.

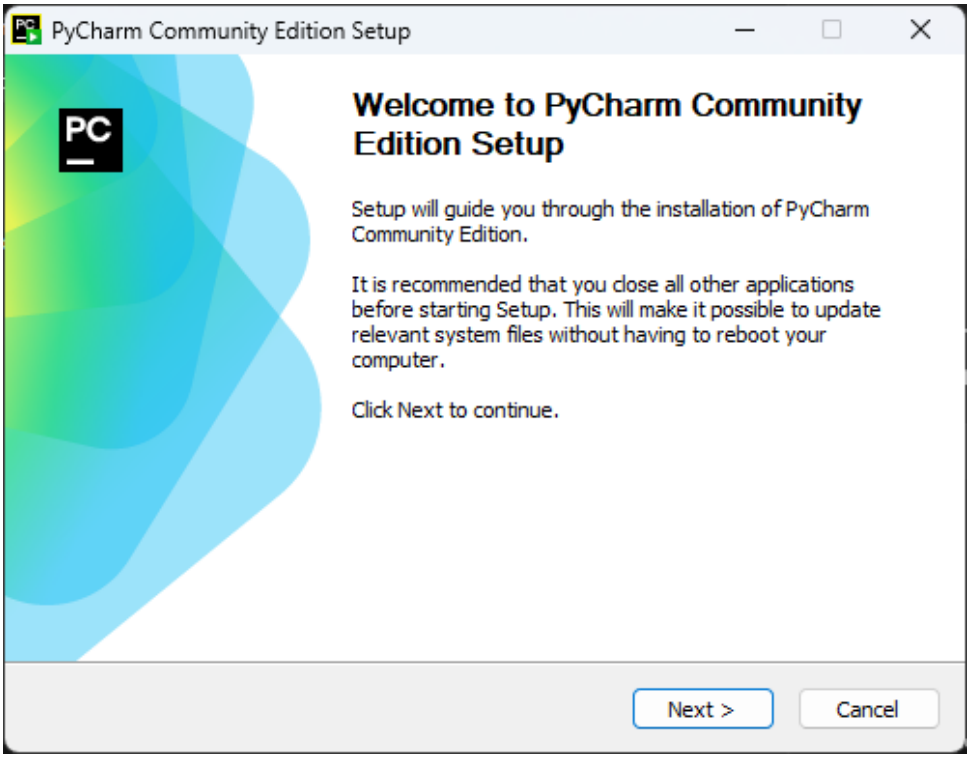

Figure 4: PyCharm Community Edition Installation Wizard

6. Once the installation is complete, you can launch PyCharm by clicking on the desktop shortcut or searching for it in the Start menu.

1.3 First Python Program

In Python, the `print()` function is used to display output on the screen. To write your first Python program, follow these steps:

1. Open PyCharm IDE on your computer.
2. Click on the "Create New Project" button to create a new project.
3. Enter a name for your project and click on the "Create" button.
4. Right-click on the project folder in the Project view and select "New" → "Python File".
5. Enter a name for your Python file and click on the "OK" button.
6. Type the following code in the Python file:

```
1    print("Hello, World!")
```

Code 1.1: Hello, World! Program

7. Click on the "Run" button to run the program.
8. You should see the output "Hello, World!" displayed in the Run window.

1.4 Basic Syntax

In Python, the syntax refers to the rules that define the combinations of symbols that are considered to be correctly structured programs in the language. The Python syntax is simple and easy to learn. It is based on indentation and does not require the use of curly braces or semicolons. It can also be easily understood by beginners as it is very similarly structured to the English language.

1.4.1 Variables

A variable is a name that refers to a value. In Python, variables are created when you assign a value to them. You can assign a value to a variable using the assignment operator `=`.

```
1  x = 5
2  y = "Hello, World!"
3
4  print(x)
5  print(y)
```

Code 1.2: Variable Assignment

Code 1.2 shows how to assign values to variables in Python. In this example, the variable `x` is assigned the value 5, and the variable `y` is assigned the value "Hello, World!".

1.4.2 Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another. In Python, an identifier is a name used to identify a variable, function, class, module, or other object. An identifier must start with a letter or an underscore (_), followed by letters, digits, or underscores.

```
# Valid Identifiers
my_variable = 5
myVariable = 10
_my_variable = 15

# Invalid Identifiers
1variable = 20
my-variable = 25
my variable = 30
```

Code 1.3: Valid and Invalid Identifiers

Code 1.3 shows examples of valid and invalid identifiers in Python. In this example, the variables `my_variable`, `myVariable`, and `_my_variable` are valid identifiers, while the variables `1variable`, `my-variable`, and `my variable` are invalid identifiers.

1.4.3 Reserved Words

Python has a set of reserved words that cannot be used as identifiers. These reserved words are used by the Python interpreter to recognize the structure of the program. Some of the reserved words in Python include:

- `False`
- `None`
- `True`
- `and`
- `as`
- `assert`
- `break`
- `class`
- `continue`
- `def`
- `del`
- `elif`
- `else`
- `except`
- `exec`
- `finally`
- `for`
- `from`
- `global`
- `if`
- `import`
- `in`
- `is`
- `lambda`
- `nonlocal`
- `not`
- `or`
- `pass`
- `print`
- `raise`
- `return`
- `try`
- `while`
- `with`
- `yield`

These reserved words cannot be used as identifiers in Python. They cannot be used as variable names, function names, or any other identifier names.

1.4.4 Quotation in Python

In Python, you can use either single quotes (’), double quotes ("), or triple quotes ("’ or """). Single and double quotes are used to represent strings, while triple quotes are used to represent multi-line strings.

```
single_quote = ’Hello, World!’
double_quote = "Hello, World!"
triple_quote = ’’’Hello,
```

```
World!'''

print(single_quote)
print(double_quote)
print(triple_quote)
```

Code 1.4: Quotation in Python

Code 1.4 shows examples of using single quotes, double quotes, and triple quotes in Python. In this example, the variables `single_quote`, `double_quote`, and `triple_quote` are assigned string values using single quotes, double quotes, and triple quotes, respectively. A single quote or double quote can be used to represent a string value, while triple quotes are used to represent multi-line strings.

1.4.4.1 Escape Characters

An escape character is a backslash (\) followed by a character that has a special meaning in Python. It is used to represent characters that are difficult or impossible to type directly. Some of the common escape characters in Python include:

- `\n` - New Line
- `\t` - Tab
- `\r` - Carriage Return
- `\\` - Backslash
- `\'` - Single Quote
- `\"` - Double Quote
- `\b` - Backspace
- `\f` - Form Feed

These escape characters are used to represent special characters in Python. For example, the escape character `\n` is used to represent a newline character, while the escape character `\t` is used to represent a tab character.

```
new_line = "Hello,\nWorld!"
tab_space = "Hello,\tWorld!"
carriage_return = "Hello,\rWorld!"
back_slash = "Hello,\\World!"
single_quote = "Hello,\'World!"
double_quote = "Hello,\"World!"
back_space = "Hello,\bWorld!"
form_feed = "Hello,\fWorld!"

print(new_line)
print(tab_space)
print(carriage_return)
print(back_slash)
print(single_quote)
print(double_quote)
print(back_space)
print(form_feed)
```

Code 1.5: Escape Characters in Python

Code 1.5 shows examples of using escape characters in Python. In this example, the variables `new_line`, `tab_space`, `back_slash`, `single_quote`, `double_quote`, `back_space`, and

`form_feed` are assigned string values using escape characters to represent special characters.

1.4.4.2 Multi-line Strings in Single or Double Quotes

In Python, you can use triple quotes (`'''` or `"""`) to represent multi-line strings. This allows you to write strings that span multiple lines without using escape characters. However, if you want to represent multi-line strings using single or double quotes, you can use the escape character `\n` to represent a newline character.

```
multi_line_single = 'Hello,\nWorld!'
multi_line_double = "Hello,\nWorld!"

print(multi_line_single)
print(multi_line_double)
```

Code 1.6: Multi-line Strings in Single or Double Quotes

Code 1.6 shows examples of using multi-line strings in single or double quotes in Python. In this example, the variables `multi_line_single` and `multi_line_double` are assigned string values using the escape character `\n` to represent a newline character.

1.4.5 Comments

Comments are used to explain the code and make it more readable. In Python, comments start with the hash character (`#`) and continue to the end of the line. Comments are ignored by the Python interpreter and are not executed as part of the program.

```
# This is a single-line comment

'''
This is a multi-line comment.
It can span multiple lines.
'''

"""
This is also a multi-line comment.
It can span multiple lines.
"""
```

Code 1.7: Comments in Python

Code 1.7 shows examples of single-line and multi-line comments in Python. In this example, the single-line comment starts with the hash character (`#`), while the multi-line comment is enclosed in triple quotes (`'''` or `"""`).

1.4.6 Lines and Indentation

Python uses indentation to define the structure of the code. Indentation is used to group statements together. The number of spaces in the indentation is not fixed, but all statements within the block must be indented the same amount.

```
if 5 > 2:
    print("Five is greater than two!")
```

Code 1.8: Lines and Indentation in Python

Code 1.8 shows an example of using indentation in Python. In this example, the `print()` statement is indented to indicate that it is part of the `if` block. The number of spaces in the indentation is not fixed, but all statements within the block must be indented the same amount.

1.4.7 Multi-Line Statements

In Python, a statement can span multiple lines if it is enclosed in parentheses `()`, square brackets `[]`, or curly braces `{}`. This is useful when you have a long statement that you want to split into multiple lines for readability.

```
numbers = [1, 2, 3, 4, 5,
           6, 7, 8, 9, 10]

total = (1 + 2 + 3 +
           4 + 5 + 6 +
           7 + 8 + 9 + 10)

colors = {'red': 255, 'green': 255,
           'blue': 255}
```

Code 1.9: Multi-Line Statements in Python

Code 1.9 shows examples of using multi-line statements in Python. In this example, the list `numbers`, the sum `total`, and the dictionary `colors` are defined using multi-line statements enclosed in square brackets, parentheses, and curly braces, respectively.

1.4.8 Data Types

In Python, every value has a data type. Data types are used to represent different types of data, such as numbers, strings, lists, tuples, dictionaries, etc. These data types are used to store, manipulate, and represent data in Python programs. Say for example, the data type `int` is used to represent integers, the data type `float` is used to represent floating-point numbers, and the data type `str` is used to represent strings. Using the correct data type is important because it determines how the data is stored and how it can be manipulated.

1.4.8.1 Integers

Integers are whole numbers, such as 1, 2, 3, 4, 5, etc. In Python, integers are represented using the `int` data type. Integers can be positive or negative, and they can be used in mathematical operations such as addition, subtraction, multiplication, and division.

```
x = 5
y = -10

```

```
print(x)
print(y)

print(type(x))
print(type(y))
```

Code 1.10: Integers in Python

Code 1.10 shows examples of using integers in Python. In this example, the variables `x` and `y` are assigned integer values 5 and -10, respectively. The `print()` function is used to display the values of the variables, and the `type()` function is used to display the data type of the variables. An integer is best used when you need to represent whole numbers without any decimal points like counting the number of students in a class or the number of books in a library.

1.4.8.2 Floats

Floats are decimal numbers, such as 1.0, 2.5, 3.14, 4.0, etc. In Python, floats are represented using the `float` data type. Floats can be used to represent real numbers, and they can be used in mathematical operations such as addition, subtraction, multiplication, and division.

```
x = 3.14
y = -2.5

print(x)
print(y)

print(type(x))
print(type(y))
```

Code 1.11: Floats in Python

Code 1.11 shows examples of using floats in Python. In this example, the variables `x` and `y` are assigned float values 3.14 and -2.5, respectively. A float is best used when you need to represent real numbers with decimal points like the price of an item or the temperature of a place.

1.4.8.3 Strings

Strings are sequences of characters, such as “Hello, World!”, “Python”, “Programming”, etc. In Python, strings are represented using the `str` data type. Strings can be enclosed in single quotes (’), double quotes ("), or triple quotes ("’ or """).

```
x = "Hello, World!"
y = 'Python'

print(x)
print(y)

print(type(x))
print(type(y))
```

Code 1.12: Strings in Python

Code 1.12 shows examples of using strings in Python. In this example, the variables `x` and `y` are assigned string values "Hello, World!" and "Python", respectively. A string is best used when you need to represent text data like names, addresses, or messages.

1.4.8.4 Lists

A list is a collection of items that are ordered and changeable. In Python, lists are represented using the `list` data type. Lists can contain items of different data types, such as integers, floats, strings, etc. Lists are enclosed in square brackets `[]` and the items are separated by commas.

```
numbers = [1, 2, 3, 4, 5]
fruits = ['apple', 'banana', 'cherry']

print(numbers)
print(fruits)

print(type(numbers))
print(type(fruits))
```

Code 1.13: Lists in Python

Code 1.13 shows examples of using lists in Python. In this example, the variables `numbers` and `fruits` are assigned list values `[1, 2, 3, 4, 5]` and `['apple', 'banana', 'cherry']`, respectively. A list is best used when you need to store multiple items in a single variable that can be changed like a list of numbers or a list of names.

1.4.8.5 Tuples

A tuple is a collection of items that are ordered and unchangeable. In Python, tuples are represented using the `tuple` data type. Tuples can contain items of different data types, such as integers, floats, strings, etc. Tuples are enclosed in parentheses `()` and the items are separated by commas.

```
coordinates = (1, 2, 3)
colors = ('red', 'green', 'blue')

print(coordinates)
print(colors)

print(type(coordinates))
print(type(colors))
```

Code 1.14: Tuples in Python

Code 1.14 shows examples of using tuples in Python. In this example, the variables `coordinates` and `colors` are assigned tuple values `(1, 2, 3)` and `('red', 'green', 'blue')`,

respectively. A tuple is best used when you need to store multiple items in a single variable that should not be changed like the coordinates of a point or the RGB values of a color.

1.4.8.6 Dictionary

A dictionary is a collection of items that are unordered, changeable, and indexed. In Python, dictionaries are represented using the `dict` data type. Dictionaries consist of key-value pairs, where each key is associated with a value. Dictionaries are enclosed in curly braces `{}` and the key-value pairs are separated by commas.

```
person = {'name': 'John', 'age': 30, 'city': 'New York'}
colors = {'red': 255, 'green': 255, 'blue': 255}

print(person)
print(colors)

print(type(person))
print(type(colors))
```

Code 1.15: Dictionaries in Python

Code 1.15 shows examples of using dictionaries in Python. In this example, the variables `person` and `colors` are assigned dictionary values `{'name': 'John', 'age': 30, 'city': 'New York'}` and `{'red': 255, 'green': 255, 'blue': 255}`, respectively. A dictionary is best used when you need to store key-value pairs like the details of a person or the RGB values of a color.

1.4.9 Conversion between Data Types

In Python, you can convert one data type to another using built-in functions. Some of the common functions used for data type conversion include:

- `int()` - Converts a value to an integer.
- `float()` - Converts a value to a float.
- `str()` - Converts a value to a string.
- `list()` - Converts a value to a list.
- `tuple()` - Converts a value to a tuple.
- `dict()` - Converts a value to a dictionary.
- `set()` - Converts a value to a set.
- `bool()` - Converts a value to a boolean.
- `chr()` - Converts an integer to a character.

```
x = 5
y = 3.14
z = '10'

list_numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1]
tuple_numbers = (1, 2, 3, 4, 5)
dict_colors = {'red': 255, 'green': 255, 'blue': 255}

print("Integer to Float:", float(x))
```

```
print("Float to Integer:", int(y))
print("String to Integer:", int(z))
print("List to Tuple:", tuple(list_numbers))
print("Tuple to List:", list(tuple_numbers))
print("Dictionary to List:", list(dict_colors))
print("List to Set:", set(list_numbers))
print("Integer to Boolean:", bool(x))
print("Integer to Character:", chr(65))
```

Code 1.16: Conversion between Data Types in Python

Code 1.16 shows examples of converting between data types in Python. In this example, the variables `x`, `y`, and `z` are assigned integer, float, and string values, respectively. The variables `list_numbers`, `tuple_numbers`, and `dict_colors` are assigned list, tuple, and dictionary values, respectively. The built-in functions are used to convert the values of these variables to different data types.

1.4.10 Basic Operators

Operators are used to perform operations on variables and values. Python supports a wide range of operators, including arithmetic operators, comparison operators, assignment operators, logical operators, etc.

1.4.10.1 Arithmetic Operators

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, etc. Some of the common arithmetic operators in Python include:

- `+` - Addition
- `-` - Subtraction
- `*` - Multiplication
- `/` - Division
- `%` - Modulus
- `**` - Exponentiation
- `//` - Floor Division

```
x = 10
y = 3

print("Addition:\t", x + y)
print("Subtraction:\t", x - y)
print("Multiplication:\t", x * y)
print("Division:\t", x / y)
print("Modulus:\t", x % y)
print("Exponentiation:\t", x ** y)
print("Floor Division:\t", x // y)
```

Code 1.17: Arithmetic Operators in Python

Code 1.17 shows examples of using arithmetic operators in Python. In this example, the variables `x` and `y` are assigned integer values 10 and 3, respectively. The arithmetic operators

are used to perform addition, subtraction, multiplication, division, modulus, exponentiation, and floor division operations on these variables.

1.4.11 PEMDAS Rule

In mathematics, the **order of operations** is a collection of rules that define the order in which different operations should be performed when evaluating an expression. The order of operations is commonly remembered using the acronym **PEMDAS**, which stands for:

- **P**arentheses
- **E**xponents
- **M**ultiplication and **D**ivision
- **A**ddition and **S**ubtraction

The PEMDAS rule is used to determine the order in which arithmetic operations should be performed in an expression. For example, in the expression $5 + 3 \times 2$, the multiplication should be performed first according to the PEMDAS rule, resulting in the value 11.

```
res1 = 5 + 3 * 2
res2 = (5 + 3) * 2

print("Result 1:", res1)
print("Result 2:", res2)
```

Code 1.18: PEMDAS Rule in Python

Code 1.18 shows examples of using the PEMDAS rule in Python. In this example, the expressions $5 + 3 \times 2$ and $(5 + 3) \times 2$ are evaluated using the arithmetic operators. The result of the first expression is 11, while the result of the second expression is 16. This demonstrates the importance of following the PEMDAS rule when evaluating arithmetic expressions.

1.4.11.1 Comparison Operators

Comparison operators are used to compare two values and determine the relationship between them. Comparison operators return a boolean value `True` or `False` based on the comparison result. Some of the common comparison operators in Python include:

- `==` - Equal
- `!=` - Not Equal
- `>` - Greater Than
- `<` - Less Than
- `>=` - Greater Than or Equal To
- `<=` - Less Than or Equal To

```
x = 10
y = 5

print("Equal:\t", x == y)
print("Not Equal:\t", x != y)
print("Greater Than:\t", x > y)
print("Less Than:\t", x < y)
print("Greater Than or Equal To:\t", x >= y)
print("Less Than or Equal To:\t", x <= y)
```

Code 1.19: Comparison Operators in Python

Code 1.19 shows examples of using comparison operators in Python. In this example, the variables `x` and `y` are assigned integer values 10 and 5, respectively. The comparison operators are used to compare the values of these variables and return boolean values `True` or `False` based on the comparison result.

1.4.11.2 Assignment Operators

Assignment operators are used to assign values to variables. They combine the assignment operator `=` with other operators to perform an operation and assign the result to a variable. Some of the common assignment operators in Python include:

- `+=` - Addition
- `-=` - Subtraction
- `*=` - Multiplication
- `/=` - Division
- `%=` - Modulus
- `**=` - Exponentiation
- `//=` - Floor Division

```
x = 10
y = 5

x += y
print("Addition:\t", x)

x -= y
print("Subtraction:\t", x)

x *= y
print("Multiplication:\t", x)

x /= y
print("Division:\t", x)

x %= y
print("Modulus:\t", x)

x **= y
print("Exponentiation:\t", x)

x //= y
print("Floor Division:\t", x)
```

Code 1.20: Assignment Operators in Python

Code 1.20 shows examples of using assignment operators in Python. In this example, the variables `x` and `y` are assigned integer values 10 and 5, respectively. The assignment operators are used to perform addition, subtraction, multiplication, division, modulus, exponentiation, and floor division operations on these variables and assign the result to the variable `x`.

1.4.11.3 Logical Operators

Logical operators are used to combine multiple conditions and determine the relationship between them. Logical operators return a boolean value `True` or `False` based on the logical relationship between the conditions. Some of the common logical operators in Python include:

- `and` - Logical AND, returns `True` if both conditions are `True`.
- `or` - Logical OR, returns `True` if at least one condition is `True`.
- `not` - Logical NOT, returns the opposite of the condition.

```
x = 10
y = 5

print("Logical AND:\t", x > 5 and y < 10)
print("Logical OR:\t", x > 5 or y > 10)
print("Logical NOT:\t", not x > 5)
```

Code 1.21: Logical Operators in Python

Code 1.21 shows examples of using logical operators in Python. In this example, the variables `x` and `y` are assigned integer values 10 and 5, respectively. The logical operators are used to combine multiple conditions and determine the relationship between them, returning boolean values `True` or `False` based on the logical relationship.

2

Control Statements

2.1 Introduction

2.2 Conditional Statements

2.2.1 If Statement

2.2.2 If-Else Statement

2.2.3 If-Elif-Else Statement

2.3 Looping Statements

2.3.1 For Loop

2.3.2 While Loop

2.3.3 Nested Loops

3

Functions

4

File Handling

4.1 Introduction

4.2 Opening and Closing Files

4.3 Reading and Writing Files

4.4 Working with Directories

4.5 CSV File

4.5.1 Reading CSV Files

4.5.2 Writing CSV Files

4.5.3 CRUD Operations in CSV Files

4.5.3.1 Create Operation

4.5.3.2 Read Operation

4.5.3.3 Update Operation

4.5.3.4 Delete Operation

5

Exception Handling

5.1 Introduction

5.2 Try-Except Block

5.3 Finally Block

5.4 Raising Exceptions

5.5 User-Defined Exceptions

6

Object-Oriented Programming

7

Modules and Packages

8

Regular Expressions

8.1 Introduction

8.2 Match Function

8.3 Search Function

8.4 Findall Function

8.5 Split Function

8.6 Sub Function

9

References

A. Books

-

B. Other Sources

-