

# Programming Fundamentals<sup>1</sup>

## A Study Guide for Students of Sorsogon State University - Bulan Campus<sup>2</sup>

JARRIAN VINCE G. GOJAR<sup>3</sup>

March 11, 2025

<sup>1</sup>A course in the Bachelor of Technical Vocational Teacher Education (BTVTED) Major in Computer System Servicing.

<sup>2</sup>This book is a study guide for students of Sorsogon State University - Bulan Campus taking up the course Programming Fundamentals.

<sup>3</sup><https://github.com/godkingjay>

Sorsogon State University - Bulan Campus

# Contents

<b>Contents</b>	<b>ii</b>
<b>List of Figures</b>	<b>v</b>
<b>List of Tables</b>	<b>vi</b>
<b>1 Introduction to Python</b>	<b>2</b>
1.1 Introduction	2
1.1.1 History of Python	2
1.1.2 What is Programming Language?	2
1.2 Environment Setup	2
1.2.1 How to Download and Install Python on Windows	2
1.2.2 How to Download PyCharm IDE on Windows	4
1.3 First Python Program	5
1.4 Basic Syntax	5
1.4.1 Variables	5
1.4.2 Python Identifiers	6
1.4.3 Reserved Words	6
1.4.4 Quotation in Python	7
1.4.4.1 Escape Characters	7
1.4.4.2 Multi-line Strings in Single or Double Quotes	8
1.4.5 Comments	8
1.4.6 Lines and Indentation	9
1.4.7 Multi-Line Statements	9
1.4.8 Data Types	10
1.4.8.1 Integers	10
1.4.8.2 Floats	10
1.4.8.3 Strings	11
1.4.8.4 Lists	12
1.4.8.5 Tuples	12
1.4.8.6 Dictionary	13
1.4.9 Conversion between Data Types	14
1.4.10 Basic Operators	15
1.4.10.1 Arithmetic Operators	15
1.4.11 PEMDAS Rule	16
1.4.11.1 Comparison Operators	16
1.4.11.2 Assignment Operators	17
1.4.11.3 Logical Operators	18
1.5 Taking Input from the User	19

<b>2</b>	<b>Control Statements</b>	<b>21</b>
2.1	Introduction . . . . .	21
2.2	Conditional Statements . . . . .	21
2.2.1	If Statement . . . . .	21
2.2.2	If-Else Statement . . . . .	22
2.2.3	If-Elif-Else Statement . . . . .	23
2.3	Iterative Statements . . . . .	24
2.3.1	While Loop . . . . .	24
2.3.2	For Loop . . . . .	25
2.3.3	For Each Loop . . . . .	26
2.3.4	Nested Loops . . . . .	27
2.4	Jump Statements . . . . .	27
2.4.1	Break Statement . . . . .	27
2.4.2	Continue Statement . . . . .	28
2.4.3	Pass Statement . . . . .	29
<b>3</b>	<b>Functions</b>	<b>30</b>
3.1	Introduction . . . . .	30
3.2	Defining Functions . . . . .	30
3.3	Calling Functions . . . . .	30
3.4	Parameters and Arguments . . . . .	31
3.5	Return Statement . . . . .	32
3.6	Recursion . . . . .	32
3.7	Lambda Functions . . . . .	33
3.8	Built-in Functions . . . . .	33
3.9	String Functions . . . . .	34
3.10	List Functions . . . . .	34
<b>4</b>	<b>File Handling</b>	<b>36</b>
4.1	Introduction . . . . .	36
4.2	Opening and Closing Files . . . . .	36
4.3	Reading and Writing Files . . . . .	36
4.4	Working with Directories . . . . .	36
4.5	CSV File . . . . .	36
4.5.1	Reading CSV Files . . . . .	36
4.5.2	Writing CSV Files . . . . .	36
4.5.3	CRUD Operations in CSV Files . . . . .	36
4.5.3.1	Create Operation . . . . .	36
4.5.3.2	Read Operation . . . . .	36
4.5.3.3	Update Operation . . . . .	36
4.5.3.4	Delete Operation . . . . .	36
<b>5</b>	<b>Exception Handling</b>	<b>37</b>
5.1	Introduction . . . . .	37
5.2	Try-Except Block . . . . .	37
5.3	Finally Block . . . . .	37
5.4	Raising Exceptions . . . . .	37
5.5	User-Defined Exceptions . . . . .	37
<b>6</b>	<b>Object-Oriented Programming</b>	<b>38</b>
6.1	Introduction . . . . .	38
6.2	Classes and Objects . . . . .	38

6.3	Inheritance . . . . .	38
6.4	Polymorphism . . . . .	38
6.5	Encapsulation . . . . .	38
6.6	Abstraction . . . . .	38
<b>7</b>	<b>Modules and Packages</b>	<b>39</b>
7.1	Introduction . . . . .	39
7.2	Creating Modules . . . . .	39
7.3	Importing Modules . . . . .	39
7.4	Creating Packages . . . . .	39
7.5	Importing Packages . . . . .	39
<b>8</b>	<b>Regular Expressions</b>	<b>40</b>
8.1	Introduction . . . . .	40
8.2	Match Function . . . . .	40
8.3	Search Function . . . . .	40
8.4	Findall Function . . . . .	40
8.5	Split Function . . . . .	40
8.6	Sub Function . . . . .	40
<b>9</b>	<b>References</b>	<b>41</b>

# List of Figures

1	Python Installation Wizard . . . . .	3
2	Python Installation Verification . . . . .	3
3	PyCharm Community Edition Download . . . . .	4
4	PyCharm Community Edition Installation Wizard . . . . .	4

# List of Tables

# List of Codes

1.1	Hello, World! Program . . . . .	5
1.2	Variable Assignment . . . . .	5
1.3	Valid and Invalid Identifiers . . . . .	6
1.4	Quotation in Python . . . . .	7
1.5	Escape Characters in Python . . . . .	8
1.6	Multi-line Strings in Single or Double Quotes . . . . .	8
1.7	Comments in Python . . . . .	8
1.8	Lines and Indentation in Python . . . . .	9
1.9	Multi-Line Statements in Python . . . . .	9
1.10	Integers in Python . . . . .	10
1.11	Floats in Python . . . . .	11
1.12	Strings in Python . . . . .	11
1.13	Lists in Python . . . . .	12
1.14	Tuples in Python . . . . .	12
1.15	Dictionaries in Python . . . . .	13
1.16	Conversion between Data Types in Python . . . . .	14
1.17	Arithmetic Operators in Python . . . . .	15
1.18	PEMDAS Rule in Python . . . . .	16
1.19	Comparison Operators in Python . . . . .	16
1.20	Assignment Operators in Python . . . . .	17
1.21	Logical Operators in Python . . . . .	18
1.22	Taking Input from the User in Python . . . . .	19
2.1	Syntax of the If Statement . . . . .	21
2.2	If Statement in Python . . . . .	21
2.3	Syntax of the If-Else Statement . . . . .	22
2.4	If-Else Statement in Python . . . . .	22
2.5	Syntax of the If-Elif-Else Statement . . . . .	23
2.6	If-Elif-Else Statement in Python . . . . .	23
2.7	Syntax of the While Loop . . . . .	24
2.8	While Loop in Python . . . . .	24
2.9	Syntax of the For Loop . . . . .	25
2.10	For Loop in Python . . . . .	25
2.11	For Loop with List in Python . . . . .	25
2.12	For Each Loop in Python . . . . .	26
2.13	Nested Loops in Python . . . . .	27
2.14	Break Statement in Python . . . . .	27
2.15	Continue Statement in Python . . . . .	28
2.16	Pass Statement in Python . . . . .	29
3.1	Defining a Function in Python . . . . .	30
3.2	Calling a Function in Python . . . . .	30
3.3	Parameters and Arguments in Python . . . . .	31



3.4	Sum of Two Numbers in Python . . . . .	31
3.5	Recursion in Python . . . . .	33
3.6	Lambda Functions in Python . . . . .	33
3.7	String Functions in Python . . . . .	34
3.8	List Functions in Python . . . . .	35

# 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 programming language. It is designed to be highly readable, using English keywords frequently, unlike other languages that use punctuation, and it has fewer syntactical constructions. It 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. A computer only understands machine code, which is a series of binary numbers. Programming languages allow humans to write code that can be understood by computers. Python in particular is an interpreted language, which means that the code is executed line by line.

## 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`.

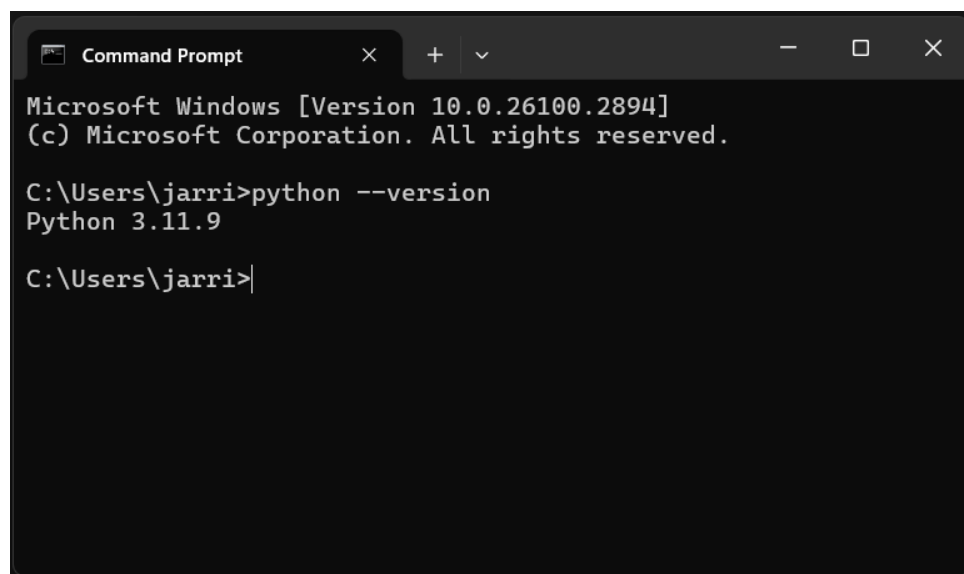


Figure 2: Python Installation Verification

### 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.

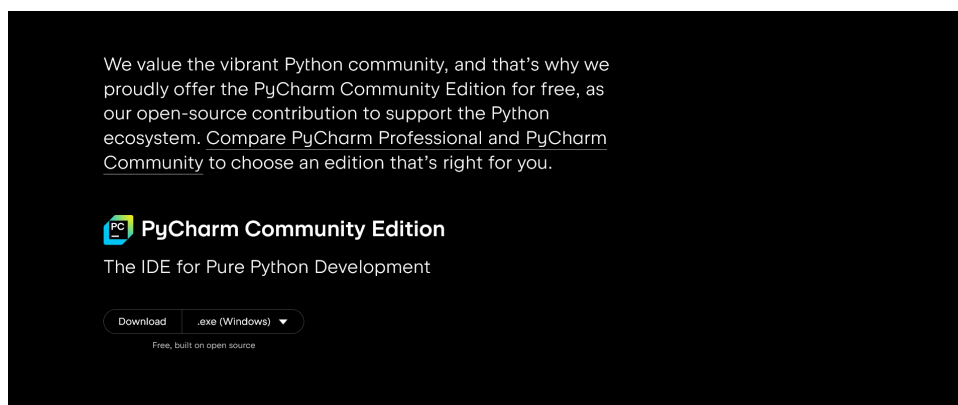


Figure 3: PyCharm Community Edition Download

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.

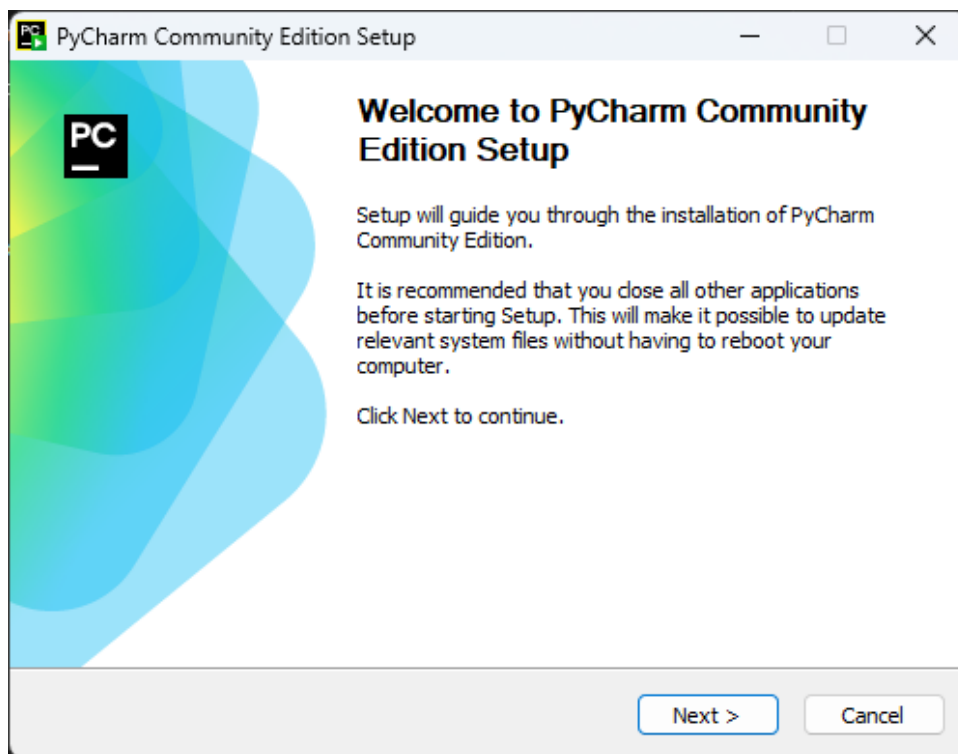


Figure 4: PyCharm Community Edition Installation Wizard

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!”.

### Exercises

1. Create a variable called 'name' and assign the value to your name.
2. Create a variable called 'age' and assign the value to your age.
3. Create a variable called 'address' and assign the value to your address.
4. Print the values of the variables 'name', 'age', and 'address'.

#### 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.

```
1 # Valid Identifiers
2 my_variable = 5
3 myVariable = 10
4 _my_variable = 15
5
6 # Invalid Identifiers
7 1variable = 20
8 my-variable = 25
9 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.

### Exercises

1. Create a variable called 'first\_name' and assign the value to your first name.
2. Create a variable called 'last\_name' and assign the value to your last name.
3. Create a variable called 'full\_name' and assign the value of concatenating 'first\_name' and 'last\_name'.
4. Print the value of the variable 'full\_name'.

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

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.

```

1 single_quote = 'Hello, World!'
2 double_quote = "Hello, World!"
3 triple_quote = '''Hello,
4 World!'''
5
6 print(single_quote)
7 print(double_quote)
8 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:

- |                                     |                                  |                               |
|-------------------------------------|----------------------------------|-------------------------------|
| • <code>\n</code> - New Line        | • <code>\\</code> - Backslash    | • <code>\b</code> - Backspace |
| • <code>\t</code> - Tab             | • <code>\'</code> - Single Quote | • <code>\f</code> - Form Feed |
| • <code>\r</code> - Carriage Return | • <code>\"</code> - Double Quote |                               |

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.



```
1 new_line = "Hello,\nWorld!"
2 tab_space = "Hello,\tWorld!"
3 carriage_return = "Hello,\rWorld!"
4 back_slash = "Hello,\\World!"
5 single_quote = "Hello,\`World!"
6 double_quote = "Hello,\"World!"
7 back_space = "Hello,\bWorld!"
8 form_feed = "Hello,\fWorld!"
9
10 print(new_line)
11 print(tab_space)
12 print(carriage_return)
13 print(back_slash)
14 print(single_quote)
15 print(double_quote)
16 print(back_space)
17 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.

```
1 multi_line_single = 'Hello,\nWorld!'
2 multi_line_double = "Hello,\nWorld!"
3
4 print(multi_line_single)
5 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.

```
1 # This is a single-line comment
2
3 '''
4 This is a multi-line comment.
5 It can span multiple lines.
6 '''
7
8 """
9 This is also a multi-line comment.
10 It can span multiple lines.
11 """
```

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.

```
1 if 5 > 2:
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.

```
1 numbers = [1, 2, 3, 4, 5,
2            6, 7, 8, 9, 10]
3
4 total = (1 + 2 + 3 +
5         4 + 5 + 6 +
6         7 + 8 + 9 + 10)
7
8 colors = {'red': 255, 'green': 255,
9          '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.

```
1 x = 5
2 y = -10
3
4 print(x)
5 print(y)
6
7 print(type(x))
8 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.

#### Exercises

1. Create a variable called 'age' and assign the value to your age.
2. Create a variable called 'year' and assign the value to the current year.
3. Create a variable called 'birth\_year' and assign the value to 0.
4. Calculate the value of the variable 'birth\_year' by subtracting 'age' from 'year'.
5. Print the value of the variable 'birth\_year'.

#### 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.

```
1 x = 3.14
2 y = -2.5
3
4 print(x)
5 print(y)
6
7 print(type(x))
8 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.

### Exercises

1. Create a variable called 'price' and assign the value to the price of a product.
2. Create a variable called 'quantity' and assign the value to the quantity of the product.
3. Create a variable called 'total' and calculate the total cost of the product by multiplying 'price' and 'quantity'.
4. Print the value of the variable 'total'.

#### 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 `"""`).

```
1 x = "Hello, World!"
2 y = 'Python'
3
4 print(x)
5 print(y)
6
7 print(type(x))
8 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.

### Exercises

1. Create a variable called 'first\_name' and assign the value to your first name.
2. Create a variable called 'last\_name' and assign the value to your last name.
3. Create a variable called 'full\_name' and concatenate 'first\_name' and 'last\_name'.
4. Print the value of the variable 'full\_name'.

#### 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.

```
1 numbers = [1, 2, 3, 4, 5]
2 fruits = ['apple', 'banana', 'cherry']
3
4 print(numbers)
5 print(fruits)
6
7 print(type(numbers))
8 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.

#### Exercises

1. Create a list called 'students' and assign the value to the names of your classmates.
2. Create a list called 'grades' and assign the value to the grades of your classmates.
3. Print the values of the lists 'students' and 'grades'.

#### 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.

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

```
3
4 print(coordinates)
5 print(colors)
6
7 print(type(coordinates))
8 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.

### Exercises

1. Create a tuple called 'point' and assign the value to the coordinates of a point.
2. Create a tuple called 'rgb' and assign the value to the RGB values of a color.
3. Print the values of the tuples 'point' and 'rgb'.

#### 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.

```
1 person = {'name': 'John', 'age': 30, 'city': 'New York'}
2 colors = {'red': 255, 'green': 255, 'blue': 255}
3
4 print(person)
5 print(colors)
6
7 print(type(person))
8 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.

```
1 x = 5
2 y = 3.14
3 z = '10'
4
5 list_numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1]
6 tuple_numbers = (1, 2, 3, 4, 5)
7 dict_colors = {'red': 255, 'green': 255, 'blue': 255}
8
9 print("Integer to Float:", float(x))
10 print("Float to Integer:", int(y))
11 print("String to Integer:", int(z))
12 print("List to Tuple:", tuple(list_numbers))
13 print("Tuple to List:", list(tuple_numbers))
14 print("Dictionary to List:", list(dict_colors))
15 print("List to Set:", set(list_numbers))
16 print("Integer to Boolean:", bool(x))
17 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.

#### Exercises

1. Create a variable called 'number' and assign the value to 10.
2. Create a variable called 'price' and assign the value to 9.99.
3. Create a variable called 'quantity' and assign the value to 5.
4. Convert the value of the variable 'number' to a `float` and assign it to a `new` variable called 'amount'.
5. Convert the value of the variable 'price' to an integer and assign it to a `new` variable called 'total'.
6. Convert the value of the variable 'quantity' to a string and assign it to

```
a new variable called 'quantity_str'.
7. Print the values of the variables 'amount', 'total', and 'quantity_str'.
```

### 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

```
1 x = 10
2 y = 3
3
4 print("Addition:\t", x + y)
5 print("Subtraction:\t", x - y)
6 print("Multiplication:\t", x * y)
7 print("Division:\t", x / y)
8 print("Modulus:\t", x % y)
9 print("Exponentiation:\t", x ** y)
10 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.

### Exercises

1. Create a variable called 'length' and assign the value to 10.
2. Create a variable called 'width' and assign the value to 5.
3. Calculate the area of a rectangle by multiplying 'length' and 'width' and assign it to a new variable called 'area'.
4. Calculate the perimeter of a rectangle by adding the sum of 'length' and 'width' and multiply it by 2 and assign it to a new variable called 'perimeter'.
5. Print the values of the variables 'area' and 'perimeter'.



### 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.

```
1 res1 = 5 + 3 * 2
2 res2 = (5 + 3) * 2
3
4 print("Result 1:", res1)
5 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.

#### Exercises

1. Create a variable called 'result1' and assign the value to  $10 + 5 * 2$ .
2. Create a variable called 'result2' and assign the value to  $(10 + 5) * 2$ .
3. Create a variable called 'result3' and assign the value to  $10 + 5 ** 2$ .
4. Create a variable called 'result4' and assign the value to  $(10 + 5) ** 2$ .
5. Print the values of the variables 'result1', 'result2', 'result3', and 'result4'.

#### 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:

- |                              |   |
|------------------------------|---|
| • <b>==</b> - Equal          | • <b>&lt;</b> - Less Than                 |
| • <b>!=</b> - Not Equal      | • <b>&gt;=</b> - Greater Than or Equal To |
| • <b>&gt;</b> - Greater Than | • <b>&lt;=</b> - Less Than or Equal To    |

```
1 x = 10
2 y = 5
```

```
3
4 print("Equal:\t", x == y)
5 print("Not Equal:\t", x != y)
6 print("Greater Than:\t", x > y)
7 print("Less Than:\t", x < y)
8 print("Greater Than or Equal To:\t", x >= y)
9 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.

### Exercises

1. Create a variable called 'age' and assign the value to your age.
2. Create a variable called 'adult' and assign the value to 18.
3. Create a variable called 'is\_adult' and compare 'age' with 'adult' using the greater than or equal to operator.
4. Print the value of the variable 'is\_adult'.

#### 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

```
1 x = 10
2 y = 5
3
4 x += y
5 print("Addition:\t", x)
6
7 x -= y
8 print("Subtraction:\t", x)
9
10 x *= y
11 print("Multiplication:\t", x)
12
13 x /= y
14 print("Division:\t", x)
15
```

```
16 x %= y
17 print("Modulus:\t", x)
18
19 x **= y
20 print("Exponentiation:\t", x)
21
22 x /= y
23 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`.

### Exercises

1. Create a variable called 'total' and assign the value to 100.
2. Create a variable called 'price' and assign the value to 10.
3. Subtract the value of the variable 'price' from the variable 'total' and assign it to the variable 'balance'.
4. Print the value of the variable 'balance'.

#### 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.

```
1 x = 10
2 y = 5
3
4 print("Logical AND:\t", x > 5 and y < 10)
5 print("Logical OR:\t", x > 5 or y > 10)
6 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.

### Exercises

1. Create a variable called 'age' and assign the value to your age.
2. Create a variable called 'adult' and assign the value to 18.
3. Create a variable called 'is\_adult' and assign the value to 'age' greater than or equal to 'adult'.
4. Create a variable called 'is\_teen' and assign the value to 'age' less than 18.
5. Print the values of the variables 'is\_adult' and 'is\_teen'.

## 1.5 Taking Input from the User

In Python, you can take input from the user using the `input()` function. The `input()` function reads a line of text from the standard input device (usually the keyboard) and returns it as a string. You can use the `input()` function to prompt the user for input and store the result in a variable.

```
1 name = input("Enter your name: ")
2 age = int(input("Enter your age: "))
3
4 print("Name:", name)
5 print("Age:", age)
```

Code 1.22: Taking Input from the User in Python

Code 1.22 shows an example of taking input from the user in Python. In this example, the `input()` function is used to prompt the user for their name and age. The input values are stored in the variables `name` and `age`, respectively. The `print()` function is used to display the values of these variables.

### Exercises

1. Greet the user.
  - Prompt the user to enter their first name and store it in a variable called 'first\_name'.
  - Prompt the user to enter their last name and store it in a variable called 'last\_name'.
  - Prompt the user to enter their age and store it in a variable called 'age'.
  - Print a message that says "Hello, <first\_name> <last\_name>! You are <age> years old."
2. Simple Calculator.
  - Prompt the user to enter two numbers and store them in variables called 'num1' and 'num2'.
  - Print the sum, difference, product, and quotient of the two numbers.
3. Favorite Color.

- Prompt the user to enter their favorite color and store it in a variable called 'color'.
- Print a message that says "<color> is a beautiful color.".

## 2

# Control Statements

## 2.1 Introduction

**Control statements** are used to control the flow of a program based on certain conditions. They allow you to make decisions, repeat actions, and perform different operations based on the conditions specified in the program. Some of the common control statements in Python include:

- **Conditional Statements** - Used to make decisions based on certain conditions.
- **Iterative Statements** - Used to repeat actions multiple times based on certain conditions.
- **Jump Statements** - Used to transfer control from one part of the program to another.

## 2.2 Conditional Statements

**Conditional statements** are used to make decisions based on certain conditions. They allow you to execute different blocks of code based on the result of a condition. Conditional statements in Python include the `if`, `if-else`, and `if-elif-else` statements.

### 2.2.1 If Statement

The `if` statement is used to execute a block of code if a condition is `True`. If the condition is `False`, the block of code is not executed. The `if` statement is written using the `if` keyword followed by the condition and a colon `:`, and the block of code is indented.

```
1 if condition:
2     # Block of code
```

Code 2.1: Syntax of the If Statement

```
1 x = int(input("Enter a number: "))
2
3 if x > 0:
4     print("The number is positive.")
```

Code 2.2: If Statement in Python

Code 2.2 shows an example of using the `if` statement in Python. In this example, the `input()` function is used to prompt the user for a number, which is stored in the variable `x`. The `if` statement is used to check if the number is positive, and if it is, a message is displayed indicating that the number is positive.

### Exercises

1. Check `if` a number is even or odd.
  - Prompt the user to enter a number and store it in a variable called `'number'`.
  - Use the `'if'` statement to check `if` the number is even or odd.
  - Print a message that says `"The number is even."` or `"The number is odd."`.
2. Check `if` a number is positive, negative, or zero.
  - Prompt the user to enter a number and store it in a variable called `'number'`.
  - Use the `'if'` statement to check `if` the number is positive, negative, or zero.
  - Print a message that says `"The number is positive."`, `"The number is negative."`, or `"The number is zero."`.
3. Check `if` a student has passed or failed.
  - Prompt the user to enter the marks of a student and store it in a variable called `'marks'`.
  - Use the `'if'` statement to check `if` the student has passed or failed.
  - Print a message that says `"The student has passed."` or `"The student has failed."`.

### 2.2.2 If-Else Statement

The `if-else` statement is used to execute one block of code if a condition is `True` and another block of code if the condition is `False`. The `if-else` statement is written using the `if` keyword followed by the condition and a colon `:`, the block of code to execute if the condition is `True`, the `else` keyword followed by a colon `:`, and the block of code to execute if the condition is `False`.

```
1 if condition:
2     # Block of code
3 else:
4     # Block of code
```

Code 2.3: Syntax of the If-Else Statement

```
1 x = int(input("Enter a number: "))
2 if x > 0:
3     print("The number is positive.")
4 else:
5     print("The number is not positive.")
```

Code 2.4: If-Else Statement in Python

Code 2.4 shows an example of using the `if-else` statement in Python. In this example, the `input()` function is used to prompt the user for a number, which is stored in the variable `x`. The `if-else` statement is used to check if the number is positive, and if it is, a message is displayed indicating that the number is positive; otherwise, a message is displayed indicating that the number is not positive.

### 2.2.3 If-Elif-Else Statement

The `if-elif-else` statement is used to execute one block of code if a condition is `True`, another block of code if another condition is `True`, and a default block of code if none of the conditions are `True`. The `if-elif-else` statement is written using the `if` keyword followed by the condition and a colon `:`, the block of code to execute if the condition is `True`, the `elif` keyword followed by the condition and a colon `:`, the block of code to execute if the condition is `True`, and the `else` keyword followed by a colon `:`, and the block of code to execute if none of the conditions are `True`.

```
1 if condition1:
2     # Block of code
3 elif condition2:
4     # Block of code
5 else:
6     # Block of code
```

Code 2.5: Syntax of the If-Elif-Else Statement

```
1 x = int(input("Enter a number: "))
2
3 if x > 0:
4     print("The number is positive.")
5 elif x < 0:
6     print("The number is negative.")
7 else:
8     print("The number is zero.")
```

Code 2.6: If-Elif-Else Statement in Python

Code 2.6 shows an example of using the `if-elif-else` statement in Python. In this example, the `input()` function is used to prompt the user for a number, which is stored in the variable `x`. The `if-elif-else` statement is used to check if the number is positive, negative, or zero, and a message is displayed based on the result of the condition.

### Exercises

1. Check `if` a number is divisible by 2, 3, or 5.
  - Prompt the user to enter a number and store it in a variable called `'number'`.
  - Use the `'if-elif-else'` statement to check `if` the number is divisible by 2, 3, or 5.
  - Print a message that says `"The number is divisible by 2."`, `"The number is divisible by 3."`, `"The number is divisible by 5."`, or `"The number is not divisible by 2, 3, or 5."`



- ```
is not divisible by 2, 3, or 5."
```
2. Check `if` a student has passed, failed, or needs improvement.
    - Prompt the user to enter the marks of a student and store it in a variable called 'marks'.
    - Use the '`if-elif-else`' statement to check `if` the student has passed, failed, or needs improvement.
    - Print a message that says "The student has passed.", "The student has failed.", or "The student needs improvement."
  3. Check `if` a number is positive, negative, or zero using the '`if-elif-else`' statement.
    - Prompt the user to enter a number and store it in a variable called 'number'.
    - Use the '`if-elif-else`' statement to check `if` the number is positive, negative, or zero.
    - Print a message that says "The number is positive.", "The number is negative.", or "The number is zero."

## 2.3 Iterative Statements

**Iterative statements** are used to repeat actions multiple times based on certain conditions. They allow you to execute the same block of code repeatedly until a condition is met. Iterative statements in Python include the `for` loop and `while` loop.

### 2.3.1 While Loop

The `while` loop is used to execute a block of code as long as a condition is `True`. The `while` loop is written using the `while` keyword followed by the condition and a colon `:`, and the block of code to execute, which is indented.

```
1 while condition:
2     # Block of code
```

Code 2.7: Syntax of the While Loop

```
1 i = 1
2 while i <= 5:
3     print(i)
4     i += 1
```

Code 2.8: While Loop in Python

Code 2.8 shows an example of using the `while` loop in Python. In this example, the variable `i` is assigned the value 1, and the `while` loop is used to print the value of `i` as long as `i` is less than or equal to 5. The value of `i` is incremented by 1 in each iteration of the loop.

### Exercises

1. Numbers from 1 to 10
  - Use the `'while'` loop to print the numbers from 1 to 10.
2. Even Numbers from 1 to 10
  - Use the `'while'` loop to print the even numbers from 1 to 10.
3. Odd Numbers from 1 to 9
  - Use the `'while'` loop to print the odd numbers from 1 to 9.

### 2.3.2 For Loop

The `for` loop is used to iterate over a sequence of items such as a list, tuple, string, etc. The `for` loop is written using the `for` keyword followed by a variable to store the items, the `in` keyword, the sequence of items, and a colon `:`, and the block of code to execute, which is indented.

```
1 for item in sequence:
2     # Block of code
```

Code 2.9: Syntax of the For Loop

```
1 for i in range(6):
2     print(i)
```

Code 2.10: For Loop in Python

Code 2.10 shows an example of using the `for` loop in Python. In this example, the `range()` function is used to generate a sequence of numbers from 0 to 5, and the `for` loop is used to iterate over these numbers and print each number in the sequence.

```
1 cars = ['Toyota', 'Honda', 'Ford']
2
3 for i in range(len(cars)):
4     print(i, cars[i])
```

Code 2.11: For Loop with List in Python

Code 2.11 shows another example of using the `for` loop in Python. In this example, the list `cars` contains three car names, and the `for` loop is used to iterate over the list and print the index and name of each car in the list.

### Exercises

1. Even Numbers from 1 to 10
  - Use the `'for'` loop to print the even numbers from 1 to 10.

**2. Iterate over a List**

- Create a list called 'colors' and assign the value to a list of colors.
- Use the 'for' loop to iterate over the colors and print each color.

**3. Multiplication Table**

- Prompt the user to enter a number and store it in a variable called 'number'.
- Use the 'for' loop to print the multiplication table of the number from 1 to 10.

Example:

Enter a number: 5

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

**2.3.3 For Each Loop**

The **for each** loop is used to iterate over a sequence of items such as a list, tuple, string, etc., and access each item in the sequence. The **for each** loop is written using the **for** keyword followed by a variable to store the items, the **in** keyword, the sequence of items, and a colon :, and the block of code to execute, which is indented.

```
1 colors = ['red', 'green', 'blue']
2
3 for color in colors:
4     print(color)
```

Code 2.12: For Each Loop in Python

Code 2.12 shows an example of using the **for each** loop in Python. In this example, the list **colors** contains three color names, and the **for each** loop is used to iterate over the list and print each color name in the list.

**Exercises****1. Characters of a String**

- Create a variable called 'name' and assign the value to your name.
- Use the 'for each' loop to iterate over the characters of the string and print each character.

### 2. Items of a List

- Create a list called 'fruit' and assign the value to a list of fruits.
- Use the 'for each' loop to iterate over the items of the list and print each item.

### 3. Pass or Fail

- Create a list called 'marks' and assign the value to a list of marks.
- Use the 'for each' loop to iterate over the marks and check if the student has passed or failed.
- Print a message that says "The student has passed." or "The student has failed."

### 4. Even or Odd

- Create a list called 'numbers' and assign the value to a list of numbers.
- Use the 'for each' loop to iterate over the numbers and check if the number is even or odd.
- Print a message that says "The number is even." or "The number is odd."

## 2.3.4 Nested Loops

**Nested loops** are loops within loops. They allow you to iterate over multiple sequences of items by placing one loop inside another loop. Nested loops are useful when you need to perform operations on multiple sequences of items simultaneously.

```
1 for i in range(3):
2     for j in range(3):
3         print(i, j)
```

Code 2.13: Nested Loops in Python

Code 2.13 shows an example of using nested loops in Python. In this example, the outer loop iterates over the numbers 0 to 2, and the inner loop iterates over the numbers 0 to 2. The `print()` function is used to display the values of `i` and `j` in each iteration of the nested loops.

## 2.4 Jump Statements

**Jump statements** are used to transfer control from one part of the program to another. They allow you to change the flow of the program based on certain conditions. Jump statements in Python include the `break`, `continue`, and `pass` statements.

### 2.4.1 Break Statement

The `break` statement is used to exit a loop prematurely based on a certain condition. When the `break` statement is encountered, the loop is terminated, and the program continues with the next statement after the loop.

```
1 for i in range(10):
2     if i == 5:
```

```
3         break
4     print(i)
```

Code 2.14: Break Statement in Python

Code 2.14 shows an example of using the `break` statement in Python. In this example, the `for` loop iterates over the numbers 0 to 9, and the `if` statement checks if the value of `i` is equal to 5. When the value of `i` is equal to 5, the `break` statement is encountered, and the loop is terminated.

### Exercises

1. Print the numbers from 1 to 10 using the `'for'` loop.
  - Use the `'for'` loop to print the numbers from 1 to 10.
  - Use the `'break'` statement to exit the loop when the number is 5.
2. Print the even numbers from 1 to 10 using the `'for'` loop.
  - Use the `'for'` loop to print the even numbers from 1 to 10.
  - Use the `'break'` statement to exit the loop when the number is 6.

### 2.4.2 Continue Statement

The `continue` statement is used to skip the current iteration of a loop based on a certain condition and continue with the next iteration. When the `continue` statement is encountered, the current iteration of the loop is skipped, and the program continues with the next iteration of the loop.

```
1 for i in range(10):
2     if i % 2 == 0:
3         continue
4     print(i)
```

Code 2.15: Continue Statement in Python

Code 2.15 shows an example of using the `continue` statement in Python. In this example, the `for` loop iterates over the numbers 0 to 9, and the `if` statement checks if the value of `i` is divisible by 2. When the value of `i` is divisible by 2, the `continue` statement is executed, and the current iteration of the loop is skipped.

### Exercises

1. Print the even numbers from 1 to 10 using the `'for'` loop.
  - Use the `'for'` loop to print the even numbers from 1 to 10.
  - Use the `'continue'` statement to skip the odd numbers.
2. Print the numbers from 1 to 10 using the `'for'` loop.
  - Use the `'for'` loop to print the numbers from 1 to 10.
  - Use the `'continue'` statement to skip the number 5.

### 2.4.3 Pass Statement

The **pass** statement is used as a placeholder when no action is required. It is often used to define empty classes, functions, or loops that need to be implemented later. The **pass** statement does nothing and allows the program to continue without raising an error.

```
1 for i in range(10):  
2     pass
```

Code 2.16: Pass Statement in Python

Code 2.16 shows an example of using the **pass** statement in Python. In this example, the **for** loop iterates over the numbers 0 to 9, and the **pass** statement is used as a placeholder inside the loop.

# 3

## Functions

### 3.1 Introduction

**Functions** are blocks of code that perform a specific task or operation. They allow you to organize your code into reusable components that can be called multiple times. Functions help you avoid writing the same code multiple times and make your code more modular and easier to maintain.

### 3.2 Defining Functions

In Python, you can define a function using the `def` keyword followed by the function name, a pair of parentheses `()`, and a colon `:`. The block of code to be executed by the function is indented.

```
1 def greet():  
2     print("Hello, World!")
```

Code 3.1: Defining a Function in Python

Code 3.1 shows an example of defining a function in Python. In this example, the `def` keyword is used to define a function called `greet`, which prints the message "Hello, World!" when called.

### 3.3 Calling Functions

To call a function in Python, you simply write the function name followed by a pair of parentheses `()`. When a function is called, the block of code inside the function is executed, and the result of the function is returned.

```
1 def greet():  
2     print("Hello, World!")  
3  
4 greet()
```

Code 3.2: Calling a Function in Python

Code 3.2 shows an example of calling a function in Python. In this example, the function `greet` is defined to print the message "Hello, World!", and the function is called to execute the block of code inside the function.

### Exercises

1. Greet the User
  - Define a function called 'greet\_user' that prints the message "Hello, User!".
  - Call the function to greet the user.
2. Print the Numbers from 1 to 10
  - Define a function called 'print\_numbers' that prints the numbers from 1 to 10 using the 'for' loop.
  - Call the function to print the numbers from 1 to 10.
3. Print the Even Numbers from 1 to 10
  - Define a function called 'print\_even\_numbers' that prints the even numbers from 1 to 10 using the 'for' loop.
  - Call the function to print the even numbers from 1 to 10.

## 3.4 Parameters and Arguments

**Parameters** are variables that are used to pass values to a function. When defining a function, you can specify one or more parameters inside the parentheses (). When calling a function, you can pass values to the parameters as **arguments**.

```
1 def greet(name):  
2     print("Hello,", name)  
3  
4 greet("Alice")
```

Code 3.3: Parameters and Arguments in Python

Code 3.3 shows an example of using parameters and arguments in Python. In this example, the function `greet` is defined with a parameter `name`, and the function is called with the argument "Alice" to pass the value to the parameter.

```
1 def add(x, y):  
2     return x + y  
3  
4 result = add(5, 3)  
5 print("Sum:", result)  
6  
7 result = add(10, 7)  
8 print("Sum:", result)
```

Code 3.4: Sum of Two Numbers in Python



Code 3.4 shows another example of using parameters and arguments in Python. In this example, the function `add` is defined with two parameters `x` and `y`, and the function is called with two arguments 5 and 3 to pass the values to the parameters.

### Exercises

#### 1. Calculator Functions

- Define a function called 'add' that takes two numbers as parameters and returns the sum of the numbers.
- Define a function called 'subtract' that takes two numbers as parameters and returns the difference of the numbers.
- Define a function called 'multiply' that takes two numbers as parameters and returns the product of the numbers.
- Define a function called 'divide' that takes two numbers as parameters and returns the quotient of the numbers.
- Print a Selection Menu that allows the user to choose an operation (add, subtract, multiply, divide).

example:

```
Selection Menu
```

1. Add
2. Subtract
3. Multiply
4. Divide

```
Enter your choice: 1
Enter the first number: 5
Enter the second number: 3
Sum: 8
```

#### 2. Greet the User with a Message

- Define a function called 'greet\_user' that takes the user's name as a parameter and prints the message "Hello, <name>!".
- Prompt the user to enter their name and store it in a variable called 'name'.
- Call the function to greet the user with the message "Hello, <name>!".

## 3.5 Return Statement

The **return** statement is a jump statement that is used to return a value from a function. When a **return** statement is encountered inside a function, the function is terminated, and the value is returned to the caller.

## 3.6 Recursion

**Recursion** is a programming technique where a function calls itself to solve a problem. Recursion is useful when a problem can be broken down into smaller subproblems that are similar to the original problem.

```
1 def factorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * factorial(n - 1)
6
7 result = factorial(5)
8 print("Factorial:", result)
```

Code 3.5: Recursion in Python

Code 3.5 shows an example of using recursion in Python. In this example, the function `factorial` is defined to calculate the factorial of a number using recursion. The function calls itself to calculate the factorial of the number `n` by multiplying `n` with the factorial of `n - 1`.

### Exercises

1. Fibonacci Series
  - Define a function called 'fibonacci' that takes a number as a parameter and returns the Fibonacci series up to that number.
  - Print the Fibonacci series up to the number 10.
2. Sum of Digits
  - Define a function called 'sum\_of\_digits' that takes a number as a parameter and returns the sum of the digits of the number.
  - Print the sum of the digits of the number 123.

## 3.7 Lambda Functions

**Lambda functions** are small, anonymous functions that can have any number of arguments but only one expression. They are defined using the `lambda` keyword followed by the arguments, a colon `:`, and the expression to be evaluated.

```
1 add = lambda x, y: x + y
2 result = add(5, 3)
3 print("Sum:", result)
```

Code 3.6: Lambda Functions in Python

Code 3.6 shows an example of using lambda functions in Python. In this example, a lambda function called `add` is defined to take two arguments `x` and `y` and return the sum of the arguments. The lambda function is called with the arguments 5 and 3 to calculate the sum of the numbers.

## 3.8 Built-in Functions

Python provides a number of built-in functions that are available for use without the need to define them. Some of the common built-in functions in Python include:

- `print()` - Used to print messages to the console.
- `input()` - Used to take input from the user.
- `len()` - Used to get the length of a sequence.
- `range()` - Used to generate a sequence of numbers.
- `sum()` - Used to calculate the sum of a sequence of numbers.
- `max()` - Used to get the maximum value in a sequence.
- `min()` - Used to get the minimum value in a sequence.

## 3.9 String Functions

Python provides a number of built-in functions that are available for working with strings. Some of the common string functions in Python include:

- `upper()` - Used to convert a string to uppercase.
- `lower()` - Used to convert a string to lowercase.
- `capitalize()` - Used to capitalize the first letter of a string.
- `title()` - Used to capitalize the first letter of each word in a string.
- `strip()` - Used to remove leading and trailing whitespace from a string.
- `replace()` - Used to replace a substring in a string with another substring.
- `split()` - Used to split a string into a list of substrings.
- `join()` - Used to join a list of strings into a single string.

```
1 name = "alice smith"
2
3 print("Uppercase:", name.upper())
4 print("Lowercase:", name.lower())
5 print("Capitalized:", name.capitalize())
6 print("Title:", name.title())
7 print("Stripped:", name.strip())
8 print("Replaced:", name.replace("a", "o"))
9 print("Split:", name.split())
10 print("Joined:", "-".join(name.split()))
```

Code 3.7: String Functions in Python

Code 3.7 shows an example of using string functions in Python. In this example, the string `name` contains the value `"alice smith"`, and various string functions are used to manipulate the string and display the results.

## 3.10 List Functions

Python provides a number of built-in functions that are available for working with lists. Some of the common list functions in Python include:

- `append()` - Used to add an item to the end of a list.
- `insert()` - Used to insert an item at a specific position in a list.
- `remove()` - Used to remove an item from a list.
- `pop()` - Used to remove and return an item from a list.
- `clear()` - Used to remove all items from a list.
- `index()` - Used to get the index of an item in a list.

- `count()` - Used to count the number of occurrences of an item in a list.
- `sort()` - Used to sort the items in a list.
- `reverse()` - Used to reverse the order of items in a list.

```
1 numbers = [5, 3, 7, 2, 8]
2
3 numbers.append(4)
4 print("Appended:", numbers)
5
6 numbers.insert(2, 6)
7 print("Inserted:", numbers)
8
9 numbers.remove(7)
10 print("Removed:", numbers)
11
12 item = numbers.pop()
13 print("Popped:", item, numbers)
14
15 numbers.clear()
16 print("Cleared:", numbers)
17
18 index = numbers.index(3)
19 print("Index:", index)
20
21 count = numbers.count(5)
22 print("Count:", count)
23
24 numbers.sort()
25 print("Sorted:", numbers)
26
27 numbers.reverse()
28 print("Reversed:", numbers)
```

Code 3.8: List Functions in Python

Code 3.8 shows an example of using list functions in Python. In this example, the list `numbers` contains the values `[5, 3, 7, 2, 8]`, and various list functions are used to manipulate the list and display the results.

# 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

6.1 Introduction

6.2 Classes and Objects

6.3 Inheritance

6.4 Polymorphism

6.5 Encapsulation

6.6 Abstraction

# 7

## Modules and Packages

7.1 Introduction

7.2 Creating Modules

7.3 Importing Modules

7.4 Creating Packages

7.5 Importing 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

-