

Class 11 Computer Science

Chapter 2: Python Programming

Posted in **11 Computer Science**

These lecture notes are prepared by **Salman Ahmad**, who is both a **teacher** and a **web developer**. The purpose of this material is to make the topic clear and useful for students of Class 10 Computer Science.



Table of Contents What is Python Programming? ● Why is Python Called Versatile? ● What is Computer Programming? ● Setting Up Python Development Environment ● Python Comments ● Variables in Python ● Types of Variables in Python ● Input Operation ● Output Operation ● Operators and Expressions in Python ● Operator Precedence in Python ● Expression in Python ● What is a Control Structure in Python? ● Main Types of Control Structures ● if Statement ● if-else Statement ● Shorthand If-Else in Python ● If-Elif-Else Statement in Python ● Looping Construct ● While Loop ● Important Programs for Exam Preparation ● For Loop ● Important Programs for Exam Preparation ● Python Modules and Built-in Data Structures ● Using Libraries and Modules in Python ● Python Built-in Data Structures: Lists ● Index Number in Python Lists ● Modifying a List ● Operations on Lists ● List Slicing ● Negative List Slicing ● Tuples in Python ● Modular Programming in Python ● Object-Oriented Programming (OOP) ● Advanced Python Concepts ● Testing and Debugging in Python ● Exercise: Short Questions ● MCQs ● MCQs for Testing ● FAQs

What is Python Programming?

Python is a high-level, easy-to-read programming language used to create software, websites, mobile applications, games, artificial intelligence (AI), data analysis applications, and many other types of programs.

Why is Python Called Versatile?

Python is called versatile because it can be used for many different types of work. Instead of learning a different programming language for every task, you can use Python for a wide variety of applications.

Web Development ● Artificial Intelligence (AI) ● Machine Learning ● Data Science ● Data Analysis ● Automation ● Game Development ● Desktop Applications ● Mobile App Development ● Cybersecurity ● Web Scraping ● Cloud Computing ● Internet of Things (IoT) ● Robotics ● Scientific Computing ● Software Development ● Networking ● Blockchain ● Image Processing ● Computer Vision

What is Computer Programming?

Computer programming is the process of writing instructions (called code) that tell a computer what to do and how to do it.

A computer cannot think or make decisions on its own. It only follows the instructions given by a programmer.

It is also called coding, programming, or software in everyday language.

Setting Up Python Development Environment

Setting up a Python development environment means installing and configuring the tools needed to write and run Python programs.

Examples of Python Development Environments

These tools are commonly called **Integrated Development Environments (IDEs)**.

Visual Studio Code (VS Code) ● PyCharm ● Google Colab (Colab) ● Jupyter Notebook ● IDLE ● Spyder

1. Compiler

A compiler translates the whole program into machine code before execution. If errors are found during compilation, they must be corrected before the program can run successfully.

Examples: C and C++ commonly use compilers.

2. Interpreter

An interpreter translates and executes program code line by line. If an error occurs, it may stop execution and show an error message.

Example: Python uses interpreter.

Python Comments

What are Comments?

Comments are notes written inside a Python program to explain the code. They are ignored by the Python interpreter, so they are not executed.

Purpose of Comments

Comments are used to:

- Explain what the code does.
- Make the code easier to read and understand.
- Help other programmers understand the program.
- Remind yourself why you wrote a particular piece of code.

Types of Comments

Python mainly has two types of comments:

1. Single-Line Comment

A single-line comment starts with the **#** symbol.

Syntax

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

Example

```
# Print a welcome message  
print("Welcome")
```

2. Multi-Line Comment

Python does not have a special multi-line comment symbol. A common way is to use triple quotes (`'''` or `"""`) to write comments that span multiple lines.

Example

```
"""
This program
prints a welcome
message.
"""
print("Welcome")
```

Variables in Python

What is a Variable?

A variable is a named location in memory that is used to store data. The value stored in a variable can be used or changed during the execution of a program.

Python Variables Example

```
x = 55
y = 66.6
name = "Salman"
age = 28
is_student = True
print(x)
print(y)
print(name)
print(age)
print(is_student)
```

Output:

```
55
66.6
Salman
28
True
```

Rules for Naming Variables

Python has some rules for naming variables.

1. Variable names must start with a letter or an underscore (`_`).

```
name = "Ali"      # Correct
_age = 20         # Correct
2name = "Ali"    # Incorrect
```

2. Variable names can contain letters, numbers, and underscores.

```
student1 = "Ahmed"    # Correct
total_marks = 450     # Correct
student-name = "Ahmed" # Incorrect (no "-")
```

3. Variable names cannot contain spaces.

```
first_name = "Ali"    # Correct
first name = "Ali"    # Incorrect
```

4. Variable names are case-sensitive. This means uppercase and lowercase letters are treated as different names.

```
name = "Ali"
Name = "Ahmed"
print(name)
print(Name)
```

Output:

```
Ali
Ahmed
```

`name` and `Name` are two different variables.

5. Do not use Python keywords as variable names.

```
if = 10      # Incorrect
class = 5    # Incorrect
```

These words are reserved by Python.

Note: Reserved words (keywords) are predefined words in Python that have a special meaning. They cannot be used as variable names, function names, or class names.

Types of Variables in Python

In Python, variables are commonly classified based on the type of value they store.

1. Integer (int)

An integer stores whole numbers (without decimal points).

Example

```
age = 20
marks = 95
```

Explanation: 20 and 95 are integers because they are whole numbers.

2. Float (float)

A float stores decimal numbers.

Example

```
height = 5.8
price = 99.99
```

Explanation: 5.8 and 99.99 are floats because they contain decimal points.

3. String (str)

A string stores text. Text is written inside single (`'`) or double (`"`) quotes.

Example

```
name = "Ali"
city = 'Lahore'
```

Explanation: "Ali" and 'Lahore' are strings because they contain text.

4. Boolean (bool)

A boolean stores only two values: **True** and **False**.

Example

```
is_student = True
is_logged_in = False
```

Explanation: True means Yes or Correct. False means No or Incorrect.

Real-Life Example

Imagine a student registration form.

```
name = "Ali"
age = 20
height = 5.8
is_student = True
```

Here: `name` → String, `age` → Integer, `height` → Float, `is_student` → Boolean.

1. Input Operation

An input operation allows the user to enter data into a program. In Python, the `input()` function is used to take input from the user.

Input an Integer (int)

Use `int(input())` to take a whole number as input.

```
age = int(input("Enter your age: "))
print(age)
```

Sample Output: Enter your age: 20
20

Input a Float (float)

Use `float(input())` to take a decimal number as input.

```
height = float(input("Enter your height: "))
print(height)
```

Sample Output: Enter your height: 5.8
5.8

Input a String (str)

Use `str(input())` to take text as input.

```
name = str(input("Enter your name: "))
print(name)
```

Sample Output: Enter your name: Ali
Ali

2. Output Operation

An output operation displays information to the user. In Python, the `print()` function is used to display output.

Syntax
`print(value)`

Example
`print("Hello, World!")`
Output: Hello, World!

Operators and Expressions in Python

What are Operators?

Operators are special symbols that perform operations on values or variables.

Example
`a = 10`
`b = 5`
`print(a + b)`
Output: 15

Here, `+` is the operator. It adds the values of `a` and `b`.

Types of Operators

1. Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations.

Operator	Meaning	Example
+	Addition	<code>10 + 5 = 15</code>
-	Subtraction	<code>10 - 5 = 5</code>
*	Multiplication	<code>10 * 5 = 50</code>
/	Division	<code>10 / 5 = 2.0</code>
%	Modulus (Remainder)	<code>10 % 3 = 1</code>
**	Exponent (Power)	<code>2 ** 3 = 8</code>
//	Floor Division	<code>10 // 3 = 3</code>

Python Arithmetic Operators Example

```
a = 10
b = 5
print("Addition:", a + b)
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Modulus:", a % 3)
print("Exponent:", 2 ** 3)
print("Floor Division:", 10 // 3)
```

Output: Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Modulus: 1
Exponent: 8
Floor Division: 3

2. Comparison (Relational) Operators

These operators compare two values and return **True** or **False**.

Operator	Meaning	Example
==	Equal to	<code>5 == 5</code> → True
!=	Not equal to	<code>5 != 3</code> → True
>	Greater than	<code>10 > 5</code> → True
<	Less than	<code>5 < 10</code> → True
>=	Greater than or equal to	<code>10 >= 10</code> → True
<=	Less than or equal to	<code>5 <= 10</code> → True

Python Comparison (Relational) Operators Example

```
a = 10
b = 5
print("Equal to:", a == b)
print("Not equal to:", a != b)
print("Greater than:", a > b)
print("Less than:", a < b)
print("Greater than or equal to:", a >= b)
print("Less than or equal to:", a <= b)
```

Output: Equal to: False
Not equal to: True
Greater than: True
Less than: False
Greater than or equal to: True
Less than or equal to: False

3. Assignment Operators

These operators assign values to variables.

Operator	Meaning	Example
=	Assign	<code>x = 10</code>
+=	Add and assign	<code>x += 5</code>
-=	Subtract and assign	<code>x -= 5</code>
*=	Multiply and assign	<code>x *= 2</code>
/=	Divide and assign	<code>x /= 2</code>
%=	Modulus and assign	<code>x %= 3</code>
//=	Floor divide and assign	<code>x //= 2</code>
**=	Exponent and assign	<code>x **= 2</code>

Python Assignment Operators Example

```
x = 10
x += 5
print(x)
x = 10
x -= 5
print(x)
x = 10
x *= 2
print(x)
x = 10
x /= 2
print(x)
x = 10
x %= 3
print(x)
x = 10
x //= 3
print(x)
x = 5
x **= 2
print(x)
```

Output: 15 1
5 3
20 25
5.0

4. Logical Operators

Logical operators combine two or more conditions into a single expression and return **True** or **False**.

Operator	Meaning
and	True if both conditions are True.
or	True if at least one condition is True.
not	Reverses the result.

Python Logical Operators Example

```
x = 10
y = 5
print(x > 5 and y < 10)
print(x > 20 or y < 10)
print(not(x > 5))
```

Output: True
True
False

Operator Precedence in Python

What is Operator Precedence?

Operator precedence is the order in which Python evaluates operators in an expression.

Simple Definition: Operator precedence is the set of rules that determines which operation is performed first in an expression.

Operator Precedence Table

Precedence	Operator	Description
1 (Highest)	()	Parentheses
2	**	Exponent
3	*, /, //, %	Multiplication, Division, Floor Division, Modulus
4	+, -	Addition, Subtraction
5	==, !=, >, <, >=, <=	Comparison Operators
6	not	Logical NOT
7	and	Logical AND
8 (Lowest)	or	Logical OR

Expression in Python

An expression is a combination of values, variables, and operators that produces a result.

Examples of Expressions in Python

```
2 + 2
2 + x
x + y
x * y + 2 - 2 + y
a - b
x * 5
a / b
x + y * 2
a + b - c
x ** 2
x % 2
(a + b) * c
x - y + z
a * b + c / 2
(x + y) * (a - b)
```

What is a Control Structure in Python?

A control structure in Python is a way of controlling the order in which a program's instructions are executed.

Main Types of Control Structures

There are mainly 3 types:

1. Sequence

Code runs from top to bottom, one statement after another.

```
print("Step 1")
print("Step 2")
print("Step 3")
```

Output: Step 1
Step 2
Step 3

Python executes the statements in order.

Real-life example: This is sequence.

2. Selection / Decision

The program makes a decision based on a condition.

Python uses: **if**, **if-else**, **if-elif-else**.

Example

```
age = 20
if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

Output: You are an adult

3. Repetition / Iteration

Sometimes we want to repeat the same code multiple times. Python uses: **for** loop, **while** loop.

Example using for loop

```
for i in range(5):
    print("Hello")
```

Output: Hello
Hello
Hello
Hello
Hello

if Statement

An if statement is a control structure used to make a decision. It executes a particular block of instructions only when a given condition is true.

```
marks = 75
if marks >= 50:
    print("Pass")
```

Output: Pass

Explanation: Since 75 >= 50 is True, the message "Pass" is displayed.

Syntax of if Statement

```
if condition:
    statement
```

if-else Statement

An if-else statement is a control structure used to make a decision between two alternatives. The if block executes when the condition is True, while the else block executes when the condition is False.

Example 1

```
marks = 40
if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

Output: Fail

Explanation: Since 40 >= 50 is False, the else block is executed and "Fail" is displayed.

Example 2

```
marks = 75
if marks >= 50:
    print("Pass")
else:
    print("Fail")
```

Output: Pass

Explanation: Since 75 >= 50 is True, the if block is executed and "Pass" is displayed.

Syntax of if-else Statement

```
if condition:
    statement
else:
    statement
```

Shorthand If-Else in Python

In Python, the shorthand form of if-else is called a ternary operator or conditional expression. It allows us to write an if-else statement in a single line.

Syntax

```
value_if_true if condition else value_if_false
```

Normal If-Else Statement

```
age = 20
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

Shorthand If-Else Statement

```
age = 20
print("Adult" if age >= 18 else "Minor")
```

Output: Adult

Explanation: If the condition `age >= 18` is True, Python prints "Adult". Otherwise, it prints "Minor".

If-Elif-Else Statement in Python

The if-elif-else statement checks multiple conditions and executes the statements in the block whose condition is true. If all conditions are false, the else block is executed.

Syntax

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    statement4
```

The following program checks a student's marks and displays the corresponding grade.

Example

```
marks = 85
if marks >= 90:
    print("A+ Grade")
elif marks >= 80:
    print("A Grade")
elif marks >= 70:
    print("B Grade")
elif marks >= 60:
    print("C Grade")
else:
    print("Fail")
```

Output: A Grade

Explanation

- **if** checks the first condition.
- **elif** checks another condition if the previous conditions are false. We can use multiple **elif** blocks.
- **else** executes when all the preceding conditions are false. It is optional.

Looping Construct

Definition: A looping construct is a control structure used to repeat a set of instructions again and again until a specific condition is met or for a specific number of times.

It is also called a repetitive structure or iterative structure.

While Loop

Definition: A while loop is a looping construct that repeats a set of instructions as long as a given condition is true.

Syntax

```
while condition:
    statement
```

Example

```
num = 1
while num <= 5:
    print(num)
    num = num + 1
```

Output: 1
2
3
4
5

How it works

1. **num** starts at 1.
2. The condition **num <= 5** is checked.
3. If it is true, the statement is executed.
4. **num** increases by 1.
5. The condition is checked again.
6. When **num** becomes 6, the condition becomes false and the loop stops.

Important Programs for Exam Preparation (While Loop)

The following programs are important from an exam point of view. Practice these programs carefully to understand the working of the while loop.

1. Program to Print Numbers from 1 to 10

```
num = 1
while num <= 10:
    print(num)
    num += 1 # num = num + 1
```

Output: 1 6
2 7
3 8
4 9
5 10

2. Program to Print Even Numbers from 1 to 20

```
i = 2
while i <= 20:
    print(i)
    i += 2 # i = i + 2
```

Output: 2 12
4 14
6 16
8 18
10 20

3. Program to Print Odd Numbers from 1 to 20

```
i = 1
while i <= 20:
    print(i)
    i += 2 # i = i + 2
```

Output: 1 11
3 13
5 15
7 17
9 19

4. Program to Print Table of a Number Stored in a Variable

```
num = 5
i = 1
while i <= 10:
    print(num, "x", i, "=", num * i)
    i += 1 # i = i + 1
```

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

5. Program to Print Table of a Number Entered by the User

```
num = int(input("Enter a number: "))
i = 1
while i <= 10:
    print(num, "x", i, "=", num * i)
    i += 1 # i = i + 1
```

Sample Output: Enter a number: 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

Note: Students must watch these program videos on my YouTube channel **ComputerWithSalman** for better understanding. This will help you understand the programs clearly and perform similar programs easily in your exams.

For Loop

Definition: A for loop is a looping construct that repeats a set of instructions for each item in a sequence or for a specific number of times.

Syntax

```
for variable in sequence:
    statement
```

Example

```
for num in range(1, 6):
    print(num)
```

Output: 1
2
3
4
5

How it works

1. **range(1, 6)** generates numbers from 1 to 5.
2. The for loop takes one number at a time.
3. The value is stored in the variable **num**.
4. The **print()** statement displays the value of **num**.
5. The loop repeats until all numbers in the sequence have been processed.

Important Programs for Exam Preparation (For Loop)

The following programs are important from an exam point of view. Practice these programs carefully to understand the working of the for loop.

1. Program to Print Numbers from 1 to 10

```
for num in range(1, 11):  
    print(num)
```

Output:

1	6
2	7
3	8
4	9
5	10

2. Program to Print Even Numbers from 1 to 20

```
for i in range(2, 21, 2):  
    print(i)
```

Output:

2	12
4	14
6	16
8	18
10	20

3. Program to Print Odd Numbers from 1 to 20

```
for i in range(1, 21, 2):  
    print(i)
```

Output:

1	11
3	13
5	15
7	17
9	19

4. Program to Print Table of a Number Stored in a Variable

```
num = 5  
for i in range(1, 11):  
    print(num, "x", i, "=", num * i)
```

Output:

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

5. Program to Print Table of a Number Entered by the User

```
num = int(input("Enter a number: "))  
for i in range(1, 11):  
    print(num, "x", i, "=", num * i)
```

Sample Output:

```
Enter a number: 7  
7 x 1 = 7  
7 x 2 = 14  
7 x 3 = 21  
7 x 4 = 28  
7 x 5 = 35  
7 x 6 = 42  
7 x 7 = 49  
7 x 8 = 56  
7 x 9 = 63  
7 x 10 = 70
```

Note: Students must watch these program videos on my YouTube channel **ComputerWithSalman** for better understanding. This will help you understand the programs clearly and perform similar programs easily in your exams.

Python Modules and Built-in Data Structures

1. Functions and Modules

Function: A function is a block of reusable code that performs a specific task. It reduces code repetition and makes programs easier to understand.

Module: A module is a Python file that contains reusable code, such as functions, variables, and classes. We can import a module to use its features in our programs.

Example

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

Output: 5.0

In this example, the `math` module provides the `sqrt()` function to calculate the square root of a number.

2. Defining and Invoking a Function

Defining a function means creating a function using the `def` keyword. Invoking a function means calling it to execute its code.

Example

```
def greet():  
    print("Hello, welcome to Python!")  
greet()
```

Output: Hello, welcome to Python!

In this example, `def greet()` defines the function, while `greet()` invokes the function and executes its code.

3. Function Parameters and Return Values

Parameters: Parameters are variables written inside a function's parentheses that receive values when the function is called.

Return Value: A return value is the result that a function sends back using the `return` statement.

Example

```
def add(a, b):  
    return a + b  
result = add(5, 3)  
print(result)
```

Output: 8

In this example, `a` and `b` are parameters. The function returns their sum, which is stored in the `result` variable.

4. Default Parameters

Default parameters are parameters that are assigned predefined values in the function definition. If no argument is provided for a parameter when calling the function, Python uses the default value specified in the parameter.

Example

```
def greet(name="Student"):  
    print("Hello,", name)  
greet()  
greet("Ali")
```

Output: Hello, Student
Hello, Ali

In this example, "Student" is the default value of the `name` parameter. When `greet()` is called without an

argument, Python uses the default value. When "Ali" is provided, the function uses that value instead.

Using Libraries and Modules in Python

1. Importing and Using Libraries

Library: A library is a collection of reusable code that helps programmers perform different tasks.

Module: A module is a Python file containing functions, variables, and classes. We can import modules to use their features without writing the code ourselves.

Example

```
import math
print(math.sqrt(36))
print(math.pi)
```

Output: 6.0
3.141592653589793

In this example, the `math` module provides the `sqrt()` function to calculate the square root of a number and `pi` to represent the value of pi.

2. Importing the random Module

The `random` module generates random values. It is useful for games, simulations, and selecting random items.

Example

```
import random
print(random.randint(1, 10))
```

Output (example): 7

The output may be different each time the program runs. The `randint(1, 10)` function generates a random integer between 1 and 10, including both numbers.

3. Importing the datetime Module

The `datetime` module is used to work with dates and times. It can display the current date and time.

Example

```
import datetime
today = datetime.date.today()
print(today)
```

Output (example): 2026-10-20

The `date.today()` function returns the current date according to the computer's local date settings.

4. Importing the statistics Module and Using the mean() Function

The `statistics` module provides functions for performing statistical calculations. Its `mean()` function calculates the average of a group of numbers.

Example

```
import statistics
marks = [70, 80, 90, 60, 100]
print(statistics.mean(marks))
```

Output: 80

In this example, the `mean()` function adds the marks and divides their total by the number of marks to calculate the average.

5. Package Structure in Python

A package is a way of organizing related Python modules into directories. It helps programmers manage large programs by grouping related code together. In simple words, a package is like a folder that groups related Python files together.

Python Built-in Data Structures: Lists

1. Lists in Python

A list is a built-in data structure used to store multiple items in a single variable. Lists can contain different data types, such as strings, integers, floats, booleans, and even other lists. They allow duplicate values and can be modified after creation. Lists are defined using square brackets `[]`.

1. List of Numbers

```
numbers = [10, 20, 30, 40, 50]
print(numbers)
```

Output: [10, 20, 30, 40, 50]

2. List of Strings

```
fruits = ["apple", "banana", "mango"]
print(fruits)
```

Output: ['apple', 'banana', 'mango']

3. List of Mixed Data Types

```
my_list = ["Ali", 25, 5.8, True]
print(my_list)
```

Output: ['Ali', 25, 5.8, True]

Accessing List Items

We can access list items using their index numbers. In Python, indexing starts from 0.

Example

```
fruits = ["Apple", "Banana", "Mango"]
print(fruits[0])
print(fruits[2])
```

Output: Apple
Mango

Index Number in Python Lists

An index number is the position of an item in a list. In Python, indexing starts from 0, not 1.

Example

```
fruits = ["apple", "banana", "mango", "orange"]
print(fruits[0])
print(fruits[1])
print(fruits[2])
print(fruits[3])
```

Output: apple
banana
mango
orange

Index Positions

Item	Index
"apple"	0
"banana"	1
"mango"	2
"orange"	3

Remember: The first item has index 0, the second has index 1, and so on. You can also use negative indexing to access items from the end of a list. For

example, `fruits[-1]` returns "orange".

Example of Negative Indexing

```
fruits = ["apple", "banana", "mango", "orange"]
print(fruits[-1])
print(fruits[-2])
```

Output: orange
mango

Modifying a List

Modifying a list means changing the elements of an existing list, such as adding, removing, or updating items after the list has been created.

Example

```
fruits = ["apple", "banana", "mango"]
fruits[1] = "orange"
print(fruits)
```

Output: ['apple', 'orange', 'mango']

Here, we changed "banana" to "orange" using its index number 1.

Operations on Lists

Python provides several methods to add, remove, and arrange list items.

1. append()

The `append()` method adds a new item to the end of a list.

Example

```
fruits = ["Apple", "Banana"]
fruits.append("Mango")
print(fruits)
```

Output: ['Apple', 'Banana', 'Mango']

2. remove()

The `remove()` method removes the first occurrence of a specified value from a list.

Example

```
fruits = ["Apple", "Banana", "Mango"]
fruits.remove("Banana")
print(fruits)
```

Output: ['Apple', 'Mango']

3. sort()

The `sort()` method arranges list items in ascending order by default.

Example

```
numbers = [40, 10, 30, 20]
numbers.sort()
print(numbers)
```

Output: [10, 20, 30, 40]

To arrange items in descending order, we can use `sort(reverse=True)`.

Example

```
numbers = [40, 10, 30, 20]
numbers.sort(reverse=True)
print(numbers)
```

Output: [40, 30, 20, 10]

4. reverse()

The `reverse()` method reverses the current order of items in a list. It does not sort the items.

Example

```
numbers = [10, 20, 30, 40]
numbers.reverse()
print(numbers)
```

Output: [40, 30, 20, 10]

List Slicing

List slicing is a technique used to access a specific portion of a list. It uses the slicing syntax `list[start:stop]`, where `start` is the starting index and `stop` is the ending index, which is not included in the result.

Syntax

`list[start:stop]`

Example

```
fruits = ["Apple", "Banana", "Mango", "Orange", "Grapes"]
print(fruits[1:4])
```

Output: ['Banana', 'Mango', 'Orange']

Explanation: Slicing starts at index 1 and stops before index 4. Therefore, the items at indexes 1, 2, and 3 are displayed.

Negative List Slicing

Negative slicing uses negative index numbers to access a portion of a list from the end. In Python, -1 represents the last item, -2 represents the second-last item, and so on.

Syntax

`list[-start:-stop]`

Example

```
fruits = ["Apple", "Banana", "Mango", "Orange", "Grapes"]
print(fruits[-4:-1])
```

Output: ['Banana', 'Mango', 'Orange']

Explanation: Slicing starts at index -4 and stops before index -1. Therefore, the items "Banana", "Mango", and "Orange" are displayed.

Understanding List Indexes

Item	Positive Index	Negative Index
Apple	0	-5
Banana	1	-4
Mango	2	-3
Orange	3	-2
Grapes	4	-1

Remember: The stop index is always excluded from the result in both positive and negative slicing.

Tuples in Python

A tuple is a built-in immutable data structure used to store multiple items in a single variable. Tuples can contain different data types, such as strings, integers, floats, booleans, and even other tuples or lists. They allow duplicate values but cannot be modified after creation. Tuples are defined using parentheses `()`.

Example

```
fruits = ("Apple", "Banana", "Mango")
print(fruits)
```

Output: ('Apple', 'Banana', 'Mango')

1. Creating a Tuple

A tuple is created by placing items inside parentheses and separating them with commas.

```
numbers = (10, 20, 30, 40, 50)
print(numbers)
```

Output: (10, 20, 30, 40, 50)

2. Accessing Tuple Items

We can access tuple items using their index numbers. Indexing starts from 0, which represents the first item.

```
fruits = ("Apple", "Banana", "Mango")
print(fruits[0])
print(fruits[2])
```

Output: Apple
Mango

3. Negative Indexing in Tuples

Negative indexing allows us to access tuple items from the end. The index -1 represents the last item, while -2 represents the second-last item.

```
fruits = ("Apple", "Banana", "Mango", "Orange")
print(fruits[-1])
print(fruits[-2])
```

Output: Orange
Mango

4. Tuple Slicing

Tuple slicing is used to access a specific portion of a tuple. It uses the syntax `tuple[start:stop]`, where `start` is the starting index and `stop` is the ending index, which is not included in the result.

```
fruits = ("Apple", "Banana", "Mango", "Orange", "Grapes")
print(fruits[1:4])
```

Output: ('Banana', 'Mango', 'Orange')

Explanation: Slicing starts at index 1 and stops before index 4. Therefore, the items at indexes 1, 2, and 3 are displayed.

5. Tuple Operations

Python provides several operations for working with tuples. One common operation is finding the number of items in a tuple.

Finding the Length Using len()

The `len()` function returns the total number of items in a tuple.

```
numbers = (10, 20, 30, 40)
print(len(numbers))
```

Output: 4

Explanation: The tuple contains four items, so the `len()` function returns 4.

Important Points About Tuples

- Tuples are defined using parentheses `()`.
- Tuples can store multiple items of different data types.
- Duplicate values are allowed in tuples.

- Tuple items can be accessed using positive and negative indexes.
- Tuple slicing allows us to access a portion of a tuple.
- Tuples are immutable, which means their items cannot be changed after creation.

Modular Programming in Python

Modular programming is a programming approach in which a large program is divided into smaller, independent parts called modules. Each module performs a specific task, making the program easier to understand, test, maintain, and reuse.

Example: Consider a calculator program. We can create separate modules for addition, subtraction, multiplication, and division. Each module performs a specific task, making the program easier to manage.

Main Function in Python

The `main()` function is used to organize the main part of a Python program. It is commonly used as the starting point for the program's main logic. In Python, `main()` is not a special or required function, but programmers use it to make their code organized and easier to understand.

Example

```
def main():
    print("Hello, World!")
    print("Welcome to Python")
```

```
if __name__ == "__main__":
    main()
```

Output: Hello, World!
Welcome to Python

Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming approach based on classes and objects. It helps organize code by combining data and functions into a single structure.

1. Class

A class is a blueprint or template used to create objects. It defines the data and functions that its objects can have.

Example: A Student class can define the name and age of a student.

2. Object

An object is an instance of a class. It represents a specific item and can use the data and functions defined in that class.

Example: Ali and Ahmed can be two different objects of the Student class.

3. Defining a Class and Creating an Object

In Python, we use the `class` keyword to define a class. We create an object by calling the class name with parentheses.

Example

```
class Student:
    def display(self):
        print("Hello, Student!")

s1 = Student()
s1.display()
```

Output: Hello, Student!

Advanced Python Concepts

1. Exception Handling

Exception handling is a method used to handle errors that occur while a program is running. It helps prevent a program from stopping suddenly when an error occurs.

Example: If a program tries to divide a number by zero, an error occurs. Exception handling allows us to handle this error properly and display a suitable message.

2. Try-Except Blocks

The try-except block is used to handle errors that may occur while a program is running. The **try** block contains code that might cause an error, while the **except** block handles the error if it occurs.

Example

```
try:
    number = 10 / 0
    print(number)
except ZeroDivisionError:
    print("Cannot divide by zero!")
```

Output: Cannot divide by zero!

Explanation

- The **try** block contains the code that may cause an error.
- Dividing 10 by 0 causes a **ZeroDivisionError**.
- The **except** block handles the error and displays a message instead of allowing the error to stop the program unexpectedly.

3. File Handling

File handling is the process of creating, opening, reading, writing, and closing files using Python. It allows programs to store information permanently and access it later.

Example: A program can save students' marks in a file and read them whenever needed.

4. Opening, Reading, Writing, and Closing a File

Python provides different functions and methods to work with files. The **open()** function opens a file, the **read()** method reads its contents, the **write()** method writes data to a file, and the **close()** method closes the file.

Example: Reading a File

```
file = open("data.txt", "r")
content = file.read()
print(content)
file.close()
```

- **open("data.txt", "r")** opens the file named data.txt in read mode.

- **file.read()** reads the contents of the file.
- **print(content)** displays the file's contents.
- **file.close()** closes the file after reading.

Example: Writing to a File

```
file = open("data.txt", "w")
file.write("Hello, Python!")
file.close()
```

- **open("data.txt", "w")** opens the file in write mode. If the file does not exist, Python creates it.
- **file.write()** writes the given text into the file.
- If the file already contains data, write mode replaces the existing content.
- **file.close()** closes the file after writing.

Common File Modes in Python

Mode	Purpose
"r"	Opens a file for reading. The file must already exist.
"w"	Opens a file for writing. Creates the file if it does not exist and replaces existing content.
"a"	Opens a file for adding data at the end without replacing existing content.

Remember: The "r" mode is used to read a file, while the "w" mode is used to write data to a file. The "a" mode is used to add new data at the end of a file.

Testing and Debugging in Python

1. Testing

Testing is the process of checking a program to make sure it works correctly and produces the expected results. It helps find errors and ensures that the program meets its requirements.

Types of Testing

1. Unit Testing — Unit testing checks a small part of a program, such as a single function, to make sure it works correctly. *Example:* Testing a function that adds two numbers.

2. Integration Testing — Integration testing checks whether different parts of a program work correctly together. *Example:* Checking whether a login form works properly with a database.

3. Functional Testing — Functional testing checks whether a program performs its required functions according to its specifications. *Example:* Checking whether a calculator correctly adds, subtracts, multiplies, and divides numbers.

4. Regression Testing — Regression testing checks whether new changes to a program have affected its existing features. *Example:* After adding a multiplication feature to a calculator, testing the addition and subtraction features again to ensure they still work correctly.

2. Debugging

Debugging is the process of finding and fixing errors in a program. It helps programmers identify why a program is not working correctly and correct the problem.

Common Debugging Techniques

1. Print Statement — The `print()` statement is used to display variable values and program results. It helps programmers check values at different stages of a program and identify where an error occurs.

Example

```
a = 10
b = 5
print("Value of a:", a)
print("Value of b:", b)
result = a + b
print("Result:", result)
```

Output: Value of a: 10
Value of b: 5
Result: 15

2. Debugging Tools — Debugging tools help programmers examine a program while it runs. They allow programmers to execute code step by step, check variable values, and find errors. *Example:* Python's built-in debugger, `pdb`, helps programmers inspect and debug their code.

3. Error Messages — Error messages provide information about problems in a program. They often show the type of error and the line where it occurred, helping programmers find and fix the problem.

Example

```
print(10 / 0)
```

Output: ZeroDivisionError: division by zero

Explanation: Python raises a `ZeroDivisionError` because a number cannot be divided by zero. The error message helps the programmer identify the problem.

Exercise: Short Questions

1. Explain the purpose of using comments in Python code.

Comments are used to explain the code and make it easier to understand. Python ignores comments during program execution. A single-line comment starts with the `#` symbol.

Example

```
# This program displays a message
print("Hello, Python!")
```

2. Describe the difference between integer and float data types in Python. Provide an example of each.

An integer (`int`) is a whole number without a decimal point, while a float (`float`) is a number that contains a decimal point.

Example

```
age = 15      # Integer
price = 25.5  # Float
```

3. Define operator precedence and give an example of an expression where operator precedence affects the result.

Operator precedence is the order in which operators are evaluated in an expression. For example, multiplication is performed before addition.

Example

```
result = 5 + 3 * 2
print(result)
```

Output: 11

Explanation: Python multiplies `3 * 2` first and then adds 5, giving the result 11.

4. How does the short-hand if-else statement differ from the regular if-else statement?

A regular if-else statement uses separate blocks of code, while a short-hand if-else statement, also called a ternary operator or conditional expression, writes the decision in a single line.

Example

```
age = 20
print("Adult" if age >= 18 else "Minor")
```

Output: Adult

5. Explain the use of the range() function in a for loop.

The `range()` function generates a sequence of numbers that is commonly used to repeat a task in a for loop. The ending value is excluded from the sequence.

Example

```
for i in range(1, 5):
    print(i)
```

Output: 1
2
3
4

6. Explain how default parameters work in Python functions.

Default parameters are function parameters that have predefined values. If an argument is not provided when calling the function, Python uses the default value.

Example

```
def greet(name="Student"):
    print("Hello,", name)
greet()
greet("Ali")
```

Output: Hello, Student
Hello, Ali

7. Explain why modular programming is useful in Python.

Modular programming divides a large program into smaller, independent parts called modules. It makes programs easier to understand, test, maintain, and reuse.

Example: A calculator program can have separate modules for addition, subtraction, multiplication, and division.

8. Explain the difference between a class and an object in Python.

A class is a blueprint or template used to create objects, while an object is an instance of a class that can use the data and methods defined in that class.

Example

```
class Student:
    def display(self):
        print("Hello, Student!")

s1 = Student()
s1.display()
```

Output: Hello, Student!

Explanation: `Student` is the class, and `s1` is the object created from that class.

MCQs

1. What is Python programming?

- a. A high-level programming language
- b. A computer hardware component