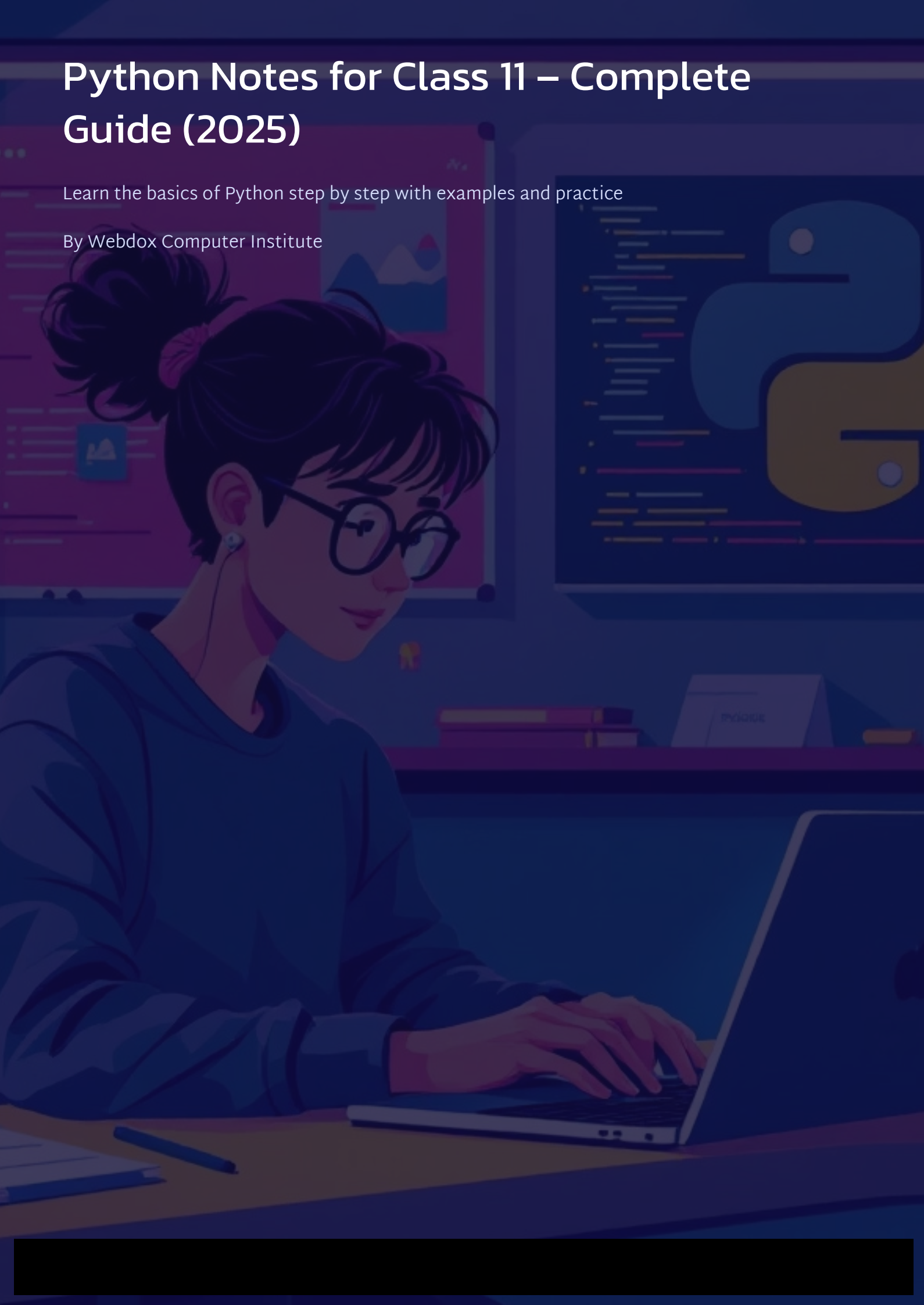
Python Notes for Class 11 – Complete Guide (2025)

Learn the basics of Python step by step with examples and practice

By Webdox Computer Institute



Introduction to Python

What is Python?

Python is a **high-level programming language** that's easy to learn and powerful to use. Created by Guido van Rossum in 1991, Python has become one of the most popular languages in the world.

Key Features

- Simple and readable syntax
- Open-source and free
- Works on all operating systems
- Huge community support
- Thousands of libraries available

Artificial Intelligence

Machine learning and AI development

Web Development

Building websites and web apps

Data Science

Analyzing and visualizing data

Automation

Automating repetitive tasks

Python Installation and Setup

01

Download Python

Visit python.org and download the latest version for your operating system (Windows, Mac, or Linux).

02

Install Python

Run the installer and make sure to check "Add Python to PATH" during installation. This allows you to run Python from anywhere on your computer.

03

Choose Your Editor

You can use IDLE (comes with Python) or install VS Code for a better coding experience. Both are great for beginners.

04

Write Your First Program

Open your editor and type your first line of code. It's that simple to get started!

Your First Python Program

```
print("Hello, Python!")
print("Welcome to Class 11 Programming!")
```

Output:

```
Hello, Python!
Welcome to Class 11 Programming!
```

Python Basics

Keywords and Identifiers

Keywords are reserved words in Python that have special meanings. Examples: `if`, `for`, `while`, `def`, `class`

Identifiers are names you give to variables, functions, and other objects. Rules:

- Must start with a letter or underscore
- Can contain letters, numbers, and underscores
- Cannot be a keyword
- Case-sensitive (age and Age are different)

Variables

Variables store data values. In Python, you don't need to declare the type – Python figures it out automatically!

```
name = "Rahul"
age = 16
height = 5.8
is_student = True
```

Data Types in Python

Integer (int)

Whole numbers

```
marks = 95
```

Float

Decimal numbers

```
price = 99.50
```

String (str)

Text in quotes

```
name = "Python"
```

Boolean (bool)

True or False

```
passed = True
```

Input and Output Functions

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))
print("Hello,", name, "! You are", age, "years old.")
```

Operators in Python

1

Arithmetic Operators

Used for mathematical calculations

- `+` Addition: 5 + 3 = 8
- `-` Subtraction: 5 - 3 = 2
- `*` Multiplication: 5 * 3 = 15
- `/` Division: 10 / 2 = 5.0
- `//` Floor Division: 10 // 3 = 3
- `%` Modulus: 10 % 3 = 1
- `**` Exponent: 2 ** 3 = 8

2

Relational Operators

Compare two values and return True or False

- `==` Equal to: 5 == 5 is True
- `!=` Not equal: 5 != 3 is True
- `>` Greater than: 5 > 3 is True
- `<` Less than: 3 < 5 is True
- `>=` Greater or equal: 5 >= 5 is True
- `<=` Less or equal: 3 <= 5 is True

3

Logical Operators

Combine multiple conditions

- `and` Both must be True
- `or` At least one must be True
- `not` Reverses the condition

```
age = 16
if age > 13 and age < 20:
    print("Teenager")
```

4

Assignment Operators

Assign values to variables

- `=` Assign: x = 5
- `+=` Add and assign: x += 3 (x = x + 3)
- `-=` Subtract and assign: x -= 2
- `*=` Multiply and assign: x *= 2
- `/=` Divide and assign: x /= 2

Conditional Statements

Conditional statements let your program make decisions based on conditions. Think of it like a flowchart where your code takes different paths based on whether something is true or false.

Basic If Statement

```
marks = 85

if marks >= 75:
    print("You got Grade A!")
    print("Excellent work!")
```

Output: You got Grade A! Excellent work!

If-Else Statement

```
age = 15

if age >= 18:
    print("You can vote!")
else:
    print("You cannot vote yet.")
```

Output: You cannot vote yet.

If-Elif-Else Statement

Use `elif` (else if) when you have multiple conditions to check.

```
marks = 82

if marks >= 90:
    grade = "A+"
elif marks >= 75:
    grade = "A"
elif marks >= 60:
    grade = "B"
elif marks >= 45:
```

Loops in Python

Loops help you repeat a block of code multiple times without writing it again and again. Python has two types of loops: `for` and `while`.

For Loop

Used when you know how many times to repeat

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

Output: 1 2 3 4 5

While Loop

Repeats as long as a condition is True

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

Output: 1 2 3 4 5

Break Statement

Stops the loop immediately

```
for i in range(1, 11):
    if i == 6:
        break
    print(i)
```

Output: 1 2 3 4 5

Continue Statement

Skips the current iteration

```
for i in range(1, 6):
    if i == 3:
        continue
    print(i)
```

Output: 1 2 4 5

Practical Example: Sum of Numbers

```
numbers = [10, 20, 30, 40, 50]
total = 0

for num in numbers:
    total += num

print("Sum:", total)
```

Strings and Lists

Working with Strings

A string is a sequence of characters enclosed in quotes. Strings are one of the most commonly used data types in Python.

```
text = "Python Programming"

# Indexing (0-based)
print(text[0])    # P
print(text[-1])   # g

# Slicing
print(text[0:6])  # Python
print(text[7:])   # Programming
```

Common String Methods

- `upper()` - Converts to uppercase
- `lower()` - Converts to lowercase
- `strip()` - Removes whitespace
- `replace()` - Replaces characters
- `split()` - Splits into a list
- `len()` - Returns length

Working with Lists

A list is a collection of items that can be of different types. Lists are ordered, changeable, and allow duplicate values.

```
fruits = ["apple", "banana", "cherry", "date"]

# Accessing elements
print(fruits[0])     # apple
print(fruits[-1])    # date
```

Tuples and Dictionaries

Tuples

A tuple is similar to a list, but it's **immutable** – once created, you cannot change its values. Tuples are faster and safer for data that shouldn't change.

```
coordinates = (10, 20, 30)
print(coordinates[0])  # 10

# This will cause an error:
# coordinates[0] = 15
```

Key Differences

- **List:** Use square brackets [ ], changeable
- **Tuple:** Use parentheses ( ), unchangeable

Dictionaries

A dictionary stores data in **key-value pairs**. Think of it like a real dictionary where you look up a word (key) to find its meaning (value).

```
student = {
    "name": "Priya",
    "age": 16,
    "grade": "A",
    "school": "DPS"
}

print(student["name"])   # Priya
print(student["grade"])  # A

# Add new entry
student["city"] = "Delhi"

# Modify existing entry
student["age"] = 17
```

Dictionary Methods

keys()
Returns all keys

```
student.keys()
```

values()
Returns all values

```
student.values()
```

items()
Returns key-value pairs

```
student.items()
```

get()
Safe way to access values

```
student.get("name")
```

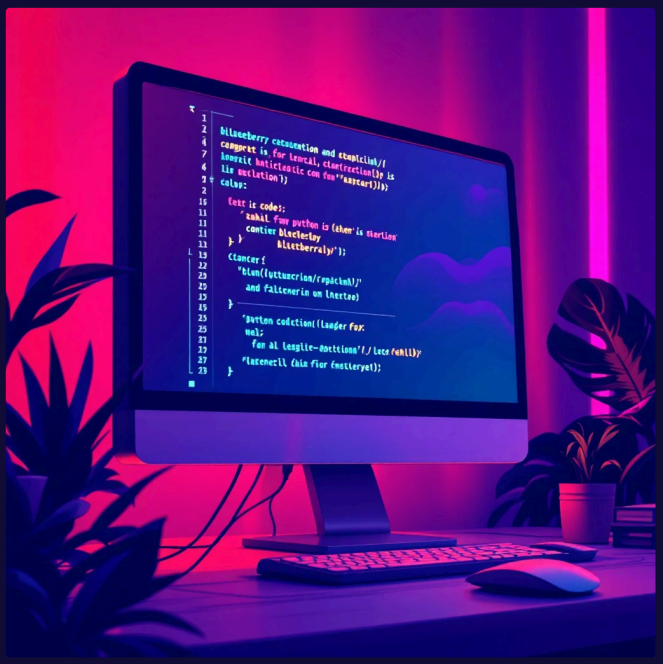
Functions, File Handling & Practice Programs

Functions in Python

A function is a reusable block of code that performs a specific task. Functions help organize code and avoid repetition.

```
def add_numbers(a, b):
    result = a + b
    return result

answer = add_numbers(10, 5)
print(answer)  # 15
```



File Handling Basics

Python can read and write files easily.

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

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

Practice Programs for Class 11 Exams

1

Find Largest Number

Check Palindrome

File Handling in Python

File handling is an essential part of any application that needs to store and retrieve data. Python makes it easy to work with files, allowing you to read data from them, write new data to them, or append existing content.

Open a File

Use the `open()` function to get a file object. This function requires the file path and the mode.

Read from a File

Once opened, use methods like `read()`, `readline()`, or `readlines()` to access the content.

Write to a File

Use the `write()` or `writelines()` method to add text to the file. Be careful with write mode as it overwrites existing content.

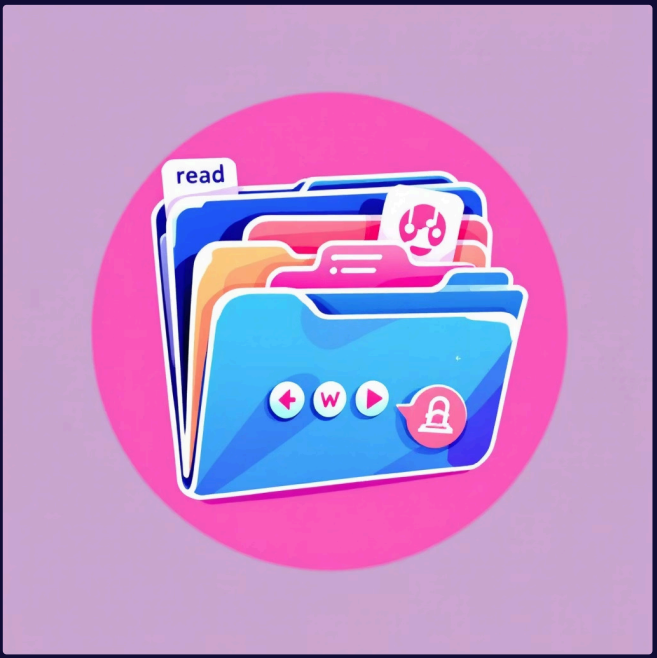
Close a File

Always close the file using `close()` to free up resources. The `with` statement is recommended for automatic closing.

File Modes

When opening a file, you specify a mode indicating how the file will be used:

- `"r"` - Read mode (default). Error if file doesn't exist.
- `"w"` - Write mode. Creates file if it doesn't exist, overwrites if it does.
- `"a"` - Append mode. Creates file if it doesn't exist, adds to end if it does.
- `"x"` - Create mode. Creates a new file, errors if it exists.
- `"t"` - Text mode (default). For text files.
- `"b"` - Binary mode. For non-text files like images or executables.



Practical Programs & Exam Preparation

To solidify your understanding of Python concepts, consistent practice is crucial. The following program types are essential for building foundational skills and preparing for your Class 11 exams. We've already covered code examples for these in previous sections; focus on understanding the logic and applying variations.

Finding Largest Number

Master comparison operators and conditional logic.

Checking Palindromes

Strengthen string manipulation and reversal techniques.

Counting Vowels/Characters

Reinforce looping through strings and character checks.

Sum of Even Numbers

Practice iteration and conditional summation within lists or ranges.

Common Errors and Debugging Tips

Understanding common errors and how to debug them is a vital skill for any programmer. Don't be discouraged by errors; view them as opportunities to learn.

Indentation Errors

Python relies heavily on consistent indentation. Incorrect spacing (e.g., mixing spaces and tabs, or inconsistent indent levels) will lead to `IndentationError`. Always use 4 spaces for indentation.

```
# Correct
if x > 0:
    print("Positive")
```

Type Errors

Occur when an operation is performed on an inappropriate data type (e.g., adding a string to an integer). Python is strongly typed, so explicit type conversion might be needed.

```
# Incorrect
name = "Alice"
age = 20
# print(name + age)
```

Logical Errors

These are the trickiest. Your code runs without crashing, but produces incorrect results. They require careful review of your algorithm, variable values, and conditional logic. Use `print()` statements or a debugger to trace execution.

```
# Example of a
logical error
```

Conclusion

As we wrap up our comprehensive guide to Python for Class 11, it's clear that you've built a robust foundation in programming. We've journeyed through the core concepts, from understanding basic data types and operators to mastering conditional statements and loops for control flow. You've learned to manipulate data structures like strings, lists, tuples, and dictionaries, and discovered the power of functions to write reusable code. Finally, we explored file handling, equipping you with the ability to manage external data effectively.

Python is more than just a subject for exams; it's a versatile tool that opens doors to countless possibilities. The skills you've acquired are highly transferable and will serve as a crucial stepping stone for future academic and professional endeavors.

Unlocking Career Opportunities

Python is a cornerstone in fields like Artificial Intelligence, Machine Learning, Data Science, Web Development, and Automation. Your foundational knowledge prepares you for advanced studies and a wide array of high-demand tech roles.

Enhancing Problem-Solving Skills

Programming inherently teaches you to break down complex problems into manageable steps. This logical thinking and structured approach are invaluable life skills, applicable far beyond coding.

Foundation for Future Innovation

The concepts learned here are universal across many programming languages. By understanding Python, you're not just learning one language, but developing a computational mindset that will allow you to adapt and innovate in a rapidly evolving technological landscape.

Continue practicing, experimenting, and building small projects. Embrace challenges, debug errors as learning opportunities, and remember that every line of code you write strengthens your understanding and capability. The world of technology awaits your contributions!