# The Complete Python Programming Book: Enhanced Notes Edition BY PYTHONMASTER

**From Basics to Advanced, With Theoretical Explanations, Concept Checklists, Quizzes & Mini Projects, approved by professionals**

---

**Table of Contents**

---

## Chapter 1: Introduction to Python

### What is Python?

Python is a high-level, interpreted, general-purpose programming language. It emphasizes code readability and simplicity, making it ideal for beginners and professionals alike.

*Key Characteristics:*

* Interpreted: No need to compile.
* Dynamically typed: You don't declare variable types explicitly.
* Multi-paradigm: Supports procedural, OOP, and functional styles.

### Why Learn Python?

* Easy syntax, similar to English.
* Huge ecosystem (Data Science, Web Dev, Automation).
* Strong community and job demand.

### Real-World Use Cases

* YouTube uses Python for video viewing
* NASA uses Python for scientific calculations
* Google uses Python for core services

### Concept Box: "Python vs Other Languages"

| Language | Typing | Use Case | Readability |
|---|---|---|---|
| Python | Dynamic | Versatile | High |
| Java | Static | Enterprise Apps | Medium |
| C | Manual | Low-level control | Low |

### Practice Set 1

1. What is Python and why is it popular?
2. List 5 features of Python.
3. Mention three areas where Python is commonly used.

---

## Chapter 2: Setting Up Python

### Installing Python

1. Visit [python.org/downloads](https://www.python.org/downloads/)
2. Install the latest version (ensure you check "Add Python to PATH").

### IDE Options

* IDLE (built-in)

* Visual Studio Code (recommended)

* Jupyter Notebook (for data science)

### Your First Program

python

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

### Running Scripts

Save as filename.py and run using:

bash

```
python filename.py
```

### Practice Set 2

1. Install Python on your machine.
2. Write a program to print your name.
3. Write a program to print the result of 2 + 3.

---

## Chapter 3: Basic Syntax

### Comments

* Single line: # This is a comment

* Multi-line: `"""This is a

  multi-line comment"""`

### Indentation (Very Important!)

python

```
if True:
    print("Correct indentation")
```

### Input/Output

python

```
name = input("Enter your name: ")
print("Hello", name)
```

### Tip:

Python uses whitespace to define blocks. Incorrect indentation will raise an error.

### Practice Set 3

1. Take user's name and greet them.
2. Write a program using correct indentation.
3. Add comments to a sample program.

---

## Chapter 4: Variables and Data Types

### What are Variables?

Variables are used to store data. Python uses dynamic typing, which means you don’t need to declare the type of a variable explicitly.

### Common Data Types

* int - Integer values (e.g. 1, 42, -7)
* float - Decimal numbers (e.g. 3.14, 2.0)
* str - Text (e.g. 'hello', '123')
* bool - Logical values (True, False)
* NoneType - Special type indicating null (None)

### Type Checking

Use type() function to check the type of a variable:

```python
x = 5
print(type(x))  # Output: <class 'int'>
```

### Type Conversion (Casting)

Convert between types using built-in functions:

```python
int("5")
float("3.14")
str(100)
bool(1)
```

### Practice Set 4

1. Create variables of different data types.
2. Print the type of each using type().
3. Convert an integer to a string and back.

---

# ## Chapter 5: Operators

### Arithmetic Operators

* + Addition
* - Subtraction
* * Multiplication
* / Division
* // Floor division
* % Modulus (remainder)
* ** Exponentiation

### Assignment Operators

* = Assign
* +=, -=, *=, etc.

### Comparison Operators

* ==, !=, >, <, >=, <=

### Logical Operators

* and, or, not

### Identity & Membership

* is, is not

* in, not in

### Bitwise Operators

Used for binary operations: &, |, ^, ~, <<, >>

### Practice Set 5

1. Try all arithmetic operations on two variables.

2. Use logical operators in conditions.

3. Check if a number is in a list using in.

---

---

> Final version will contain all 35 chapters in the enhanced format with theory, examples, syntax, practice sets, and tips.

# Chapter 6: Control Flow (if, elif, else)

What is Control Flow?

Control flow lets your program make decisions by executing certain parts of the code based on conditions.

Syntax:

if condition:

```
    # block of code
elif condition:
    # block of code
else:
    # block of code
```

**Example:**

```
x = 10
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

**Nested Conditions**

**You can nest if statements within other if blocks.**

**Concept Checklist ✅**

- **Understand conditional logic**
- **Syntax of if-elif-else**
- **Use of comparison/logical operators in conditions**

**Practice Set 6**

1. **Check if a number is even or odd**
2. **Check if a number is positive, negative, or zero**
3. **Find the largest of three numbers**

**Quiz ✍️**

1. **What keyword is used for fallback logic?**
2. **Which operator checks equality?**

**Mini Project 💡**

**Simple Grading System: Input student marks and print grades:**

- **= 90: A**
- **80–89: B**
- **70–79: C**

- <70: Fail

---

# Chapter 7: Loops (for, while)

**Why Use Loops?**

**Loops let you repeat a block of code multiple times. Great for automation and iteration.**

**Types of Loops**

- for loop
- while loop

**For Loop Example:**

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

**While Loop Example:**

```
count = 0
while count < 5:
    print(count)
    count += 1
```

**Loop Control**

- break: exit loop
- continue: skip to next iteration
- pass: placeholder

**Concept Checklist ✅**

- Use for with range()
- Understand while loop condition
- Apply break/continue correctly

**Practice Set 7**

1. Print numbers from 1 to 10
2. Find factorial of a number using loop
3. Print even numbers between 1 and 20

**Quiz ✍**

1. What does range(3, 7) return?
2. Which loop is used when number of iterations is unknown?

**Mini Project 💡**

**Prime Number Checker: Accept a number and check if it is prime using a loop**

---

# Chapter 8: Functions

**What is a Function?**

**A function is a block of reusable code that performs a specific task. It allows for better modularity, easier testing, and avoids repetition.**

**Why Use Functions?**

- **Makes code organized and readable**
- **Encourages code reuse**
- **Simplifies debugging**

**Defining a Function:**

```
def greet(name):
    print("Hello", name)
```

**Calling a Function:**

```
greet("Alice")
```

**Return Statement:**

**Use return to send back results:**

```
def add(a, b):
    return a + b
```

**Types of Function Arguments:**

- **Positional Arguments – Based on position**
- **Keyword Arguments – Based on parameter names**
- **Default Arguments – Predefined default values**
- **Variable-Length Arguments – *args, **kwargs**

**Example:**

```
def student(name, age=18):
    print(name, age)


student("Alice")          # Uses default age
student("Bob", 21)        # Overrides default age
```

**Variable Scope:**

- **Local Scope – Exists within a function**
- **Global Scope – Defined outside all functions**

**Concept Checklist ✅**

- **Define and call functions**
- **Understand and use return statements**
- **Differentiate between argument types**
- **Understand variable scope**

**Practice Set 8**

1. **Create a function to calculate the square of a number**
2. **Write a calculator using separate functions for each operation**
3. **Demonstrate use of *args and **kwargs**

**Quiz ✍**

1. **What does the return keyword do?**
2. **What is the difference between *args and **kwargs?**
3. **Can a function have no return value?**

**Mini Project 💡**

**Grocery Bill Calculator:**

- **Input item names and prices**
- **Calculate total, tax, and final bill**
- **Use functions for subtotal, tax, and final calculation**

---

# Chapter 9: Data Structures – Lists

**What is a List?**

**A list is a mutable, ordered collection that can hold a mix of data types.**

**Creating a List:**

**fruits = ["apple", "banana", "cherry"]**

**mixed = [1, "hello", 3.5, True]**

**Common List Operations:**

- **Access: fruits[0]**
- **Update: fruits[1] = "blueberry"**

- Append: fruits.append("grape")
- Remove: fruits.remove("banana")
- Insert: fruits.insert(1, "kiwi")
- Sort/Reverse: fruits.sort(), fruits.reverse()

Slicing:

```
print(fruits[1:3])
```

Looping:

```
for item in fruits:
    print(item)
```

Concept Checklist ✅

- Create and manipulate lists
- Access and modify elements
- Understand slicing and looping

Practice Set 9

1. Create a list and find the sum of all elements
2. Sort a list of strings alphabetically
3. Remove a specific item and print the new list

Quiz ✍️

1. What method adds a new item at the end?
2. What is the index of the last element?
3. Are lists mutable or immutable?

Mini Project 💡

To-Do List Manager:

- Add, view, and remove tasks using list operations
- Display pending task count

---

# Chapter 10: Data Structures – Tuples

What is a Tuple?

A tuple is an ordered, immutable collection used to store fixed data.

Creating a Tuple:

```
coordinates = (10, 20)
```

**mixed = (1, "yes", 3.5)**

**Accessing Elements:**

**print(coordinates[0])**

**Tuple Unpacking:**

**x, y = coordinates**

**Immutability:**

**Tuples cannot be changed after creation.**

**Concept Checklist ✅**

- **Define and access tuple elements**
- **Understand tuple immutability**
- **Use unpacking effectively**

**Practice Set 10**

1. **Create a tuple with five elements**
2. **Print the second and fourth element**
3. **Try to modify a tuple and note the error**

**Quiz ✍️**

1. **Can tuples be sorted?**
2. **How do you unpack a tuple?**
3. **What makes tuples faster than lists?**

**Mini Project 💡**

**Geographic Coordinates Saver:**

- **Store multiple (lat, long) tuples**
- **Access and print coordinates by index**

---

# Chapter 11: Data Structures – Sets

**What is a Set?**

**A set is an unordered, mutable collection of unique items.**

**Creating a Set:**

**nums = {1, 2, 3, 3, 2}  # duplicates removed**

**Set Operations:**

**A = {1, 2, 3}**

**B = {3, 4, 5}**

**print(A | B)  # union**

**print(A & B)  # intersection**

**print(A - B)  # difference**

**Methods:**

- **add(), remove(), discard(), clear()**

**Concept Checklist ✅**

- **Understand set uniqueness**
- **Perform union, intersection, and difference**
- **Add/remove elements**

**Practice Set 11**

1. **Create two sets and perform all set operations**
2. **Check if a value exists in a set**
3. **Convert a list to a set to remove duplicates**

**Quiz ✍️**

1. **Are sets ordered?**
2. **What method removes an element without error?**
3. **What's the result of {1, 2} | {2, 3}?**

**Mini Project 💡**

**Duplicate Filter:**

- **Input a list with duplicate items**
- **Convert to set to show only unique items**

---

# Chapter 12: Data Structures – Dictionaries

**What is a Dictionary?**

**A dictionary stores key-value pairs. It's mutable and unordered (Python 3.6+ maintains insertion order).**

**Creating a Dictionary:**

**person = {"name": "John", "age": 30}**

**Accessing and Modifying:**

**print(person["name"])**

```
person["age"] = 31
```

**Dictionary Methods:**

- keys(), values(), items()
- get(), update(), pop()

**Looping:**

```
for key, value in person.items():
    print(key, value)
```

**Concept Checklist ✅**

- Create and access dictionary items
- Use keys, values, items
- Loop through dictionary

**Practice Set 12**

1. Create a dictionary of student marks
2. Update and delete a key
3. Loop through dictionary to print values

**Quiz ✍️**

1. What method returns all keys?
2. How do you safely access a key?
3. What data type are dictionary keys?

**Mini Project 💡**

**Phone Book App:**

- Store contact name and number
- Search, update, and delete contacts

---

# Chapter 13: String Handling

**What is a String?**

**A string is a sequence of characters enclosed in single, double, or triple quotes.**

**String Operations:**

```
text = "Hello, Python!"
print(text.upper())
print(text.lower())
```

```
print(text.replace("Python", "World"))
```

**Slicing and Indexing:**

```
print(text[0:5])  # Hello
print(text[-1])   # !
```

**Common Methods:**

- upper(), lower(), title(), replace()
- find(), split(), join()

**f-Strings:**

```
name = "Alice"
print(f"Hello, {name}!")
```

**Concept Checklist ✅**

- Index and slice strings
- Use built-in string methods
- Use formatted strings (f-strings)

**Practice Set 13**

1. Concatenate two strings
2. Count the vowels in a string
3. Reverse a string using slicing

**Quiz ✍️**

1. What does text.split() do?
2. How do you insert a variable inside a string?
3. Are strings mutable?

**Mini Project 💡**

**Text Formatter:**

- Input paragraph
- Count word frequency
- Highlight long words

---

**Enhanced Python Notes – Continuation (Chapters 14 to 18)**

---

# Chapter 14: File Handling

**What is File Handling?**

**File handling allows your program to create, read, write, and modify files stored on disk. This is useful for saving user input, processing logs, or reading configurations.**

**File Modes:**

- **'r' – Read (default)**
- **'w' – Write (overwrites existing file)**
- **'a' – Append to existing file**
- **'b' – Binary mode**
- **'x' – Create new file (error if file exists)**

**Opening and Closing Files:**

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

**Using with Statement (Recommended):**

```
with open("example.txt", "r") as file:
    data = file.read()
# File is automatically closed
```

**Writing to Files:**

```
with open("output.txt", "w") as f:
    f.write("Hello, world!\n")
```

**Reading Line by Line:**

```
with open("example.txt", "r") as file:
    for line in file:
        print(line)
```

**Concept Checklist ✅**

- 

**Practice Set 14**

1. **Write your name and age to a file**
2. **Read a file and print each line**
3. **Count the number of words in a file**

**Quiz ✍**

1. **What does mode 'a' do?**

2. Why use the with statement?
3. Which method reads all lines as a list?

Mini Project 💡

Note Taker App:

- Append daily notes with timestamps to a file
- View past notes on demand

---

(Chapters 15 to 35 will follow this structure...)

# Chapter 15: Exception Handling

What is an Exception?

An exception is an error that occurs during program execution, disrupting the normal flow of the program.

Why Handle Exceptions?

To prevent your program from crashing and handle errors gracefully.

Try-Except Block:

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

Catching Multiple Exceptions:

```
try:
    number = int("abc")
except ValueError:
    print("Invalid input!")
```

Else and Finally:

```
try:
    print("No error")
except:
    print("An error occurred")
else:
    print("This runs if no error occurs")
```

**finally:**

```
    print("Always runs")
```

**Raising Custom Exceptions:**

```
raise ValueError("This is a custom error")
```

**Concept Checklist ✅**

- 

**Practice Set 15**

1. Handle a division by zero error
2. Catch a type conversion error
3. Use finally to always close a file

**Quiz ✍️**

1. What does finally do?
2. What happens if no error occurs in try block?
3. What keyword is used to raise custom errors?

**Mini Project 💡**

**Robust Calculator:**

- Ask user for two numbers and an operation
- Use exception handling for invalid inputs (e.g. divide by zero, invalid operation)

---

# Chapter 16: Object-Oriented Programming (OOP)

**What is OOP?**

**Object-Oriented Programming is a paradigm based on the concept of "objects" which bundle data and functionality.**

**Class and Object:**

```
class Person:
    def __init__(self, name):
        self.name = name


p = Person("Alice")
print(p.name)
```

**Key Concepts:**

- **Encapsulation – Hiding data inside classes**
- **Inheritance – A class inherits properties from another**
- **Polymorphism – One method behaves differently based on object**
- **Abstraction – Hiding internal logic, showing only necessary details**

**Inheritance:**

```
class Animal:
    def speak(self):
        print("Animal speaks")


class Dog(Animal):
    def speak(self):
        print("Dog barks")
```

**Private Members:**

```
class Secret:
    def __init__(self):
        self.__code = "1234"  # private variable
```

**Concept Checklist ✅**

- 

**Practice Set 16**

1. **Create a class with attributes and methods**
2. **Inherit a class and override its method**
3. **Use __init__, self, and private variables**

**Quiz ✍️**

1. **What is __init__() used for?**
2. **How do you make an attribute private?**
3. **What is method overriding?**

**Mini Project 💡**

**Library System:**

- **Create a class for Book**
- **Store title, author, and availability**
- **Inherit to create BorrowedBook with due date**

# Chapter 17: Modules and Packages

**What is a Module?**

**A module is a file containing Python code (functions, classes, etc.) that can be reused in other scripts.**

**Using Built-in Modules:**

```
import math

print(math.sqrt(16))
```

**Creating Your Own Module:**

**mymodule.py**

```
def greet(name):

    return f"Hello, {name}!"
```

**Using it:**

```
import mymodule

print(mymodule.greet("Alice"))
```

**What is a Package?**

**A package is a directory of modules with a special __init__.py file.**

**from-import:**

```
from math import pi

print(pi)
```

**Concept Checklist ✅**

- 

**Practice Set 17**

1. **Use math and random modules**
2. **Create a custom module with 2 functions**
3. **Import only specific functions**

**Quiz ✍️**

1. **What is the purpose of __init__.py?**
2. **How do you import a function directly?**
3. **What's the difference between module and package?**

**Mini Project 💡**

**Utility Toolkit:**

- **Create a module with date/time utilities**
- **Import and use it in a main program**

---

# Chapter 18: Working with Libraries

**What is a Library?**

**A library is a collection of modules intended to help with specific tasks like data analysis, web scraping, etc.**

**Common Built-in Libraries:**

- **math: math functions**
- **random: random numbers**
- **datetime: time/date manipulation**

**Installing External Libraries:**

```
pip install requests
```

**Using an Installed Library:**

```
import requests

response = requests.get("https://api.github.com")

print(response.status_code)
```

**Concept Checklist ✅**

- 

**Practice Set 18**

1. **Use random to simulate a coin toss**
2. **Get today's date using datetime**
3. **Install and use the requests module**

**Quiz ✍️**

1. **What is pip?**
2. **How do you install a library?**
3. **Name a library used for web requests**

**Mini Project 💡**

**Dice Simulator:**

- **Use random to simulate dice rolls**
- **Allow user to roll repeatedly**

---

**Enhancement for the remaining chapters is now underway in this new continuation document.**

**Enhanced Python Notes – Continuation (Chapters 19 to 35)**

---

# Chapter 19: Regular Expressions

**What is a Regular Expression (Regex)?**

A regular expression is a special sequence of characters used to match patterns in text. It is extremely useful for validation, searching, and text processing.

**Importing the re Module:**

import re

**Common Regex Functions:**

- re.search() – Search for a match
- re.match() – Match from the start of a string
- re.findall() – Find all matches
- re.sub() – Replace text using pattern

**Basic Patterns:**

- . – Any character
- ^ – Start of string
- $ – End of string
- * – 0 or more
- + – 1 or more
- ? – 0 or 1
- \d – Digit
- \w – Word character

**Example:**

pattern = r"\d+"

text = "There are 42 apples"

result = re.findall(pattern, text)

```
print(result)  # ['42']
```

**Concept Checklist ✅**

- Use re module to search and match text
- Understand common regex symbols
- Extract and replace patterns in strings

**Practice Set 19**

1. Validate an email address using regex
2. Extract all phone numbers from a paragraph
3. Replace all digits in a string with #

**Quiz ✍️**

1. What does \d match?
2. Which function returns all matches?
3. What is the difference between match() and search()?

**Mini Project 💡**

**Form Validator**:

- Input user email, phone, and name
- Validate each using regex before accepting

---

(Chapters 20 to 35 will continue in this format...)

# Chapter 20: Comprehensions

**What is a Comprehension?**

Comprehension is a concise way to create new sequences like lists, sets, and dictionaries using a single line of code with a loop and condition.

**List Comprehension:**

```
squares = [x**2 for x in range(10)]
```

**Conditional Comprehension:**

```
evens = [x for x in range(10) if x % 2 == 0]
```

**Set Comprehension:**

```
unique = {char for char in "banana"}
```

**Dictionary Comprehension:**

```
double = {x: x*2 for x in range(5)}
```

**Concept Checklist** ✅

- Use list, set, and dictionary comprehensions
- Add conditions inside comprehensions

**Practice Set 20**

1. Create a list of cubes for numbers 1–10
2. Generate a set of vowels in a string
3. Build a dictionary of items and their lengths from a list of strings

**Quiz** ✍️

1. What does [x for x in range(3)] output?
2. How do you add a condition to a comprehension?
3. What structure does {x: x*x for x in range(3)} return?

**Mini Project** 💡

**Student Grader**:

- Given a dictionary of names and marks
- Create a new dictionary with names and grade "Pass"/"Fail" based on marks using dict comprehension

---

# Chapter 21: Lambda, Map, Filter, Reduce

**What is a Lambda Function?**

Lambda functions are small, anonymous functions defined using the lambda keyword.

```
square = lambda x: x * x
print(square(5))
```

**map():**

Applies a function to each item in a list.

```
nums = [1, 2, 3]
squares = list(map(lambda x: x**2, nums))
```

**filter():**

Returns items for which the function returns True.

```
evens = list(filter(lambda x: x % 2 == 0, nums))
```

**reduce():**

Reduces a list to a single value (needs functools).

```
from functools import reduce

product = reduce(lambda x, y: x * y, nums)
```

**Concept Checklist ✅**

- Define and use lambda functions
- Use map(), filter(), and reduce() correctly

**Practice Set 21**

1. Use map to convert temperature from Celsius to Fahrenheit
2. Use filter to remove negative numbers from a list
3. Use reduce to find the sum of a list

**Quiz ✍️**

1. What does map() return?
2. Which module provides reduce()?
3. Can lambda functions have multiple expressions?

**Mini Project 💡**

**Discount Calculator**:

- Use map to apply discount to product prices
- Use filter to show discounted items above ₹100

---

# Chapter 22: Decorators

**What is a Decorator?**

A decorator is a function that takes another function and extends or modifies its behavior without changing the original function code.

**Creating and Using a Decorator:**

```
def my_decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper
```

```
@my_decorator
def greet():
    print("Hello!")


greet()
```

**Concept Checklist** ✅

- Understand decorator syntax
- Modify function behavior using decorators

**Practice Set 22**

1. Create a decorator that logs the time a function was called
2. Create a decorator that prints the arguments passed to a function
3. Nest two decorators

**Quiz** ✍️

1. What symbol is used to apply a decorator?
2. Can decorators be reused?
3. What is a wrapper function?

**Mini Project** 💡

**Authentication Decorator**:

- A decorator checks if user is logged in before running the target function

---

# Chapter 23: Iterators and Generators

**Iterators:**

Objects that can be iterated one element at a time using next().

```
nums = iter([1, 2, 3])
print(next(nums))
```

**Generators:**

Functions that yield items one at a time using yield.

```
def countdown(n):
    while n > 0:
        yield n
        n -= 1
```

```
for i in countdown(5):
    print(i)
```

**Generator Expressions:**

```
squares = (x**2 for x in range(5))
```

**Concept Checklist ✅**

- Understand iterators vs. generators
- Use iter() and next()
- Create functions using yield

**Practice Set 23**

1. Create a generator that yields even numbers up to 20
2. Convert a list to an iterator and manually loop using next()
3. Use generator expression to generate cube of numbers

**Quiz ✍️**

1. What does yield do?
2. How is a generator different from a normal function?
3. Are generator expressions memory efficient?

**Mini Project 💡**

**Lazy Fibonacci Generator**:

- Create a generator that yields Fibonacci numbers up to N terms

---

# Chapter 24: Context Managers

**What is a Context Manager?**

A context manager is used to manage resources like files or database connections. It ensures that resources are properly cleaned up after use.

**Using with Statement:**

```
with open("file.txt") as f:
    data = f.read()
```

**Custom Context Manager:**

```
class MyContext:
    def __enter__(self):
```

```
        print("Entering")
        return self
    def __exit__(self, exc_type, exc_val, exc_tb):
        print("Exiting")

with MyContext():
    print("Inside block")
```

**Concept Checklist** ✅

- Use with statement for resource management
- Implement custom context managers with __enter__ and __exit__

**Practice Set 24**

1. Read and write to a file using with
2. Create a custom context manager for opening a connection (mock)
3. Add error handling in __exit__

**Quiz** ✍️

1. What are the two methods every context manager must define?
2. What happens if an error occurs inside with block?
3. Can context managers be nested?

**Mini Project** 💡

**File Logger**:

- Create a context manager that logs file access and errors

---

# Chapter 25: Debugging and Testing

**Why Debug and Test?**

Debugging helps identify and fix errors; testing ensures your program behaves as expected.

**Types of Errors:**

- **Syntax Errors** – Mistakes in code structure
- **Runtime Errors** – Errors during execution
- **Logic Errors** – Code runs but output is incorrect

**Debugging Techniques:**

- Print statements

- IDE debuggers (breakpoints)
- Using pdb (Python debugger)

```
import pdb; pdb.set_trace()
```

**Unit Testing with unittest:**

```
import unittest


def add(x, y):
    return x + y


class TestAdd(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2, 3), 5)


unittest.main()
```

**Concept Checklist** ✅

- Understand different error types
- Use pdb and breakpoints to debug
- Write test cases using unittest

**Practice Set 25**

1. Use print/debugger to find a bug in a loop
2. Create a test case for a calculator function
3. Handle a logic error by rewriting code

**Quiz** ✍️

1. What are the three error types?
2. What module is used for debugging?
3. What is a unit test?

**Mini Project** 💡

**Bug Tracker**:

- Accept user input
- Log errors and test common functions

# Chapter 26: Multithreading & Multiprocessing

**Why Use Threads and Processes?**

To run multiple tasks concurrently, improving performance for I/O and CPU-bound tasks.

**Threading:**

```
import threading

def greet():
    print("Hello")

thread = threading.Thread(target=greet)
thread.start()
```

**Multiprocessing:**

```
from multiprocessing import Process

def greet():
    print("Hello")

p = Process(target=greet)
p.start()
```

**Concept Checklist ☑**

- Use threads for concurrent I/O tasks
- Use processes for CPU-heavy tasks

**Practice Set 26**

1. Create a thread that prints numbers 1–5
2. Create a process that multiplies two numbers
3. Compare runtime of single vs. multi-threaded code

**Quiz ✍**

1. Which is better for CPU-bound tasks: threading or multiprocessing?
2. Which module starts a new process?

3. What function runs a thread?

**Mini Project 💡**

**Parallel Downloader**:

- Download multiple files simultaneously using threads

---

# Chapter 27: Working with Databases (SQLite3)

**What is SQLite?**

SQLite is a lightweight, embedded database engine used for storing structured data.

**Connecting to a Database:**

```
import sqlite3
conn = sqlite3.connect("mydb.db")
cursor = conn.cursor()
```

**Creating and Using Tables:**

```
cursor.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER, name TEXT)")
cursor.execute("INSERT INTO users VALUES (1, 'Alice')")
conn.commit()
```

**Fetching Data:**

```
cursor.execute("SELECT * FROM users")
print(cursor.fetchall())
```

**Concept Checklist ✅**

- Connect to SQLite DB
- Create tables and insert data
- Query and fetch results

**Practice Set 27**

1. Create a table for products
2. Insert and retrieve data using SQL
3. Update and delete a record

**Quiz ✍**

1. What function connects to SQLite?
2. How do you execute an SQL query?
3. Why call commit()?

**Mini Project 💡**

**Simple Address Book**:

- Add, view, update, delete contacts using SQLite

---

# Chapter 28: Web Scraping with BeautifulSoup

**What is Web Scraping?**

Extracting data from websites programmatically.

**Installing & Importing:**

```
pip install beautifulsoup4 requests
import requests
from bs4 import BeautifulSoup
```

**Basic Scraping:**

```
res = requests.get("https://example.com")
soup = BeautifulSoup(res.text, "html.parser")
print(soup.title.text)
```

**Navigating Elements:**

```
soup.find("h1")
soup.find_all("a")
```

**Concept Checklist ✅**

- Send requests using requests
- Parse HTML using BeautifulSoup
- Navigate and extract tags/attributes

**Practice Set 28**

1. Extract all links from a webpage
2. Get all headings from a page
3. Scrape prices from an online product page

**Quiz ✍**

1. What is BeautifulSoup used for?
2. How do you parse a webpage?
3. Which module sends HTTP requests?

**Mini Project 💡**

**News Headline Aggregator**:

- Scrape headlines from a news site and display

---

# Chapter 29: APIs and Requests

**What is an API?**

An API (Application Programming Interface) allows different software systems to communicate.

**Sending Requests:**

```
import requests
res = requests.get("https://api.github.com")
print(res.status_code)
print(res.json())
```

**Common HTTP Methods:**

- GET – Retrieve data
- POST – Submit data
- PUT – Update data
- DELETE – Remove data

**Parameters and Headers:**

```
res = requests.get("https://api.example.com", params={"q": "python"})
```

**Concept Checklist ✅**

- Use requests to make API calls
- Handle response codes
- Pass headers and parameters

**Practice Set 29**

1. Send a GET request to a public API
2. Display weather data using an API
3. Submit a POST request

**Quiz ✍**

1. What does requests.get() return?
2. What does res.json() do?
3. What are status codes 200 and 404?

**Mini Project 💡**

**Weather Dashboard**:

- Fetch and display weather info for a city using an API

---

# Chapter 30: GUI with Tkinter

**What is Tkinter?**

Tkinter is Python's standard GUI (Graphical User Interface) library. It helps create window-based applications.

**Basic Window:**

```
import tkinter as tk


root = tk.Tk()
root.title("My App")
root.mainloop()
```

**Adding Widgets:**

```
label = tk.Label(root, text="Hello")
label.pack()


button = tk.Button(root, text="Click Me", command=lambda: print("Clicked"))
button.pack()
```

**Concept Checklist** ✅

- Create a basic Tkinter window
- Use labels, buttons, entry widgets
- Bind events using command

**Practice Set 30**

1. Create a window with a title and button
2. Add an entry box that prints the value on button click
3. Add a label that updates with input text

**Quiz** ✍

1. What method keeps the GUI running?
2. Which widget displays text?
3. What argument is used for click event?

**Mini Project** 💡

**Simple To-Do App**:

- Add tasks using entry
- Display tasks in a listbox
- Delete tasks with a button

---

# Chapter 31: Web Dev with Flask

**What is Flask?**

Flask is a micro web framework for building web applications in Python.

**Installing and Running:**

```
pip install flask
from flask import Flask
app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome!"

app.run(debug=True)
```

**Templates:**

Use Jinja2 with render_template() to serve HTML pages.

**Concept Checklist** ☑

- Create and run Flask app
- Define routes
- Return HTML using templates

**Practice Set 31**

1. Create a Flask app with 2 routes
2. Return an HTML file with a welcome message
3. Handle user input using query strings

**Quiz** ✍

1. What does @app.route() do?

2. What port does Flask run on by default?
3. What is Jinja2 used for?

**Mini Project 💡**

**Feedback Form Website**:

- Collect name, email, and feedback
- Show submitted data on new page

---

# Chapter 32: Data Analysis with Pandas

**What is Pandas?**

Pandas is a powerful data manipulation library. It uses Series and DataFrame for tabular data.

**Installing:**

```
pip install pandas
```

**Reading Data:**

```
import pandas as pd
df = pd.read_csv("data.csv")
print(df.head())
```

**Basic Operations:**

```
print(df.columns)
print(df["Age"].mean())
```

**Concept Checklist ✅**

- Create and load DataFrames
- Perform column/row operations
- Summarize and filter data

**Practice Set 32**

1. Load CSV and print shape
2. Display rows with condition Age > 25
3. Add new column based on existing ones

**Quiz ✍**

1. What is a DataFrame?
2. What does df.head() return?
3. How do you filter rows?

**Mini Project 💡**

**Student Report Analyzer**:

- Read marks from CSV
- Compute average and grade for each student

---

**Chapter 33: Data Visualization (Matplotlib & Seaborn)**

**Why Visualize Data?**

Helps understand trends, patterns, and outliers.

**Matplotlib:**

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 5, 6])
plt.title("Line Graph")
plt.show()
```

**Seaborn:**

```
import seaborn as sns
sns.barplot(x="name", y="score", data=df)
plt.show()
```

**Concept Checklist ✅**

- Plot graphs using Matplotlib
- Create charts with Seaborn
- Label axes, add titles, legends

**Practice Set 33**

1. Create a line chart using plt.plot()
2. Visualize student scores with sns.barplot
3. Plot correlation heatmap

**Quiz ✍️**

1. What does plt.show() do?
2. What's a Seaborn function to create bar graphs?
3. Can Seaborn work with Pandas?

**Mini Project 💡**

**Dashboard Visualizer**:

- Use Seaborn to chart user stats and Matplotlib to track changes over time

---

# Chapter 34: Intro to Machine Learning

**What is Machine Learning (ML)?**

ML is the ability for systems to learn from data and improve without being explicitly programmed.

**Types of ML:**

- **Supervised Learning** – Labeled data
- **Unsupervised Learning** – Unlabeled data

**Libraries:**

- Scikit-learn
- Pandas
- Numpy

**Example:**

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y)
```

**Concept Checklist** ✅

- Understand supervised vs. unsupervised learning
- Use Scikit-learn for basic modeling
- Preprocess data

**Practice Set 34**

1. Train a simple linear regression model
2. Use train-test split
3. Predict and evaluate accuracy

**Quiz** ✍️

1. What is fit() in ML?
2. What library provides ML models?
3. What is training vs. testing data?

**Mini Project** 💡

**Salary Predictor**:

- Use linear regression to predict salary from experience

# Chapter 35: Advanced Python Tips & Tricks

**Useful Features:**

- List unpacking: a, b = [1, 2]
- Chained comparisons: 1 < x < 10
- Ternary operator: result = x if condition else y
- Zip and enumerate:

```
for i, value in enumerate(mylist):
    print(i, value)
```

**Memory Optimization:**

- Use __slots__ in classes
- Use generator expressions over lists

**Performance Tips:**

- Use built-in functions (sum, min, max)
- Avoid unnecessary loops and recursion

**Concept Checklist ✅**

- Use concise Python idioms
- Write memory and performance-efficient code

**Practice Set 35**

1. Use zip to pair two lists
2. Use ternary operation for grading logic
3. Create a class with __slots__

**Quiz ✍**

1. What is __slots__ used for?
2. What does enumerate() return?
3. How to optimize loops in Python?

**Mini Project 💡**

**Efficiency Analyzer**:

- Profile and optimize a given Python script

🎉 **Congratulations!** You've completed all 35 enhanced chapters of Python programming. This guide now serves as a full, video-free reference with all theory, quizzes, and mini projects.

WRITTEN BY :

M GIRISH

(Professional developer )