Complete Python Programming Book

From Beginner to Advanced

Table of Contents

Chapter 1: Introduction to Python {#introduction}

What is Python?

Python is a high-level, interpreted programming language created by Guido van Rossum in 1991. It emphasizes code readability and simplicity, making it an excellent choice for beginners and professionals alike.

Why Learn Python?

- **Easy to Learn**: Clear, readable syntax
- **Versatile**: Web development, data science, AI, automation, and more
- **Large Community**: Extensive libraries and frameworks
- **In-Demand**: High job market demand
- **Cross-Platform**: Works on Windows, Mac, and Linux

Python Philosophy

The Zen of Python (PEP 20) guides Python's design:

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Readability counts

Chapter 2: Getting Started {#getting-started}

Installing Python

1. **Download Python**: Visit python.org and download the latest version
2. **Install**: Run the installer (check "Add Python to PATH")
3. **Verify Installation**: Open terminal/command prompt and type:

```bash
python --version
```

Your First Python Program

Create a file called `hello.py`:

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

Run it:

```bash
python hello.py
```

Python Interactive Mode

Start the Python interpreter:

```bash
python
```

Try some commands:

```python
>>> 2 + 2
4
>>> print("Python is fun!")
Python is fun!
>>> exit()
```

IDEs and Text Editors

Popular choices:

- **VS Code**: Free, powerful, extensive plugins
- **PyCharm**: Professional IDE with many features
- **Jupyter Notebook**: Great for data science
- **IDLE**: Comes with Python

Chapter 3: Basic Data Types {#basic-data-types}

Numbers

Python has three numeric types:

Integers (int)

```python
age = 25
population = 7_900_000_000  # Underscores for readability
negative = -42
```

Floating-point (float)

```python
pi = 3.14159
temperature = -15.5
scientific = 1.23e-4  # Scientific notation
```

Complex Numbers

```python
complex_num = 3 + 4j
```

Strings (str)

Strings are sequences of characters:

python

```python
# Single quotes
name = 'Alice'

# Double quotes
message = "Hello, World!"

# Triple quotes for multiline
poem = """Roses are red,
Violets are blue,
Python is awesome,
And so are you!"""

# String operations
greeting = "Hello"
name = "Bob"
full_greeting = greeting + ", " + name + "!"  # Concatenation

# String methods
text = "Python Programming"
print(text.upper())      # PYTHON PROGRAMMING
print(text.lower())      # python programming
print(text.replace("Python", "Java"))  # Java Programming
print(len(text))         # 18
```

String Formatting

python

```python
# f-strings (Python 3.6+)
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old")

# .format() method
print("My name is {} and I am {} years old".format(name, age))

# % formatting (older style)
print("My name is %s and I am %d years old" % (name, age))
```

Booleans (bool)

python

```python
is_student = True
is_working = False

# Boolean operations
print(True and False)  # False
print(True or False)   # True
print(not True)        # False
```

None Type

python

```python
result = None  # Represents absence of value
```

Type Conversion

python

```python
# Convert between types
num_str = "123"
num_int = int(num_str)      # 123
num_float = float(num_str)  # 123.0

age = 25
age_str = str(age)          # "25"

# Check type
print(type(42))             # <class 'int'>
print(type("Hello"))        # <class 'str'>
print(isinstance(42, int))  # True
```

Chapter 4: Operators and Expressions {#operators-expressions}

Arithmetic Operators

python

```python
a = 10
b = 3

print(a + b)   # 13 - Addition
print(a - b)   # 7  - Subtraction
print(a * b)   # 30 - Multiplication
print(a / b)   # 3.333... - Division
print(a // b)  # 3  - Floor division
print(a % b)   # 1  - Modulo (remainder)
print(a ** b)  # 1000 - Exponentiation
```

Comparison Operators

python

```python
x = 5
y = 10

print(x == y)  # False - Equal to
print(x != y)  # True  - Not equal to
print(x < y)   # True  - Less than
print(x > y)   # False - Greater than
print(x <= y)  # True  - Less than or equal to
print(x >= y)  # False - Greater than or equal to
```

Logical Operators

python

```python
a = True
b = False

print(a and b)  # False
print(a or b)   # True
print(not a)    # False

# Practical example
age = 25
has_license = True
can_rent_car = age >= 21 and has_license  # True
```

Assignment Operators

python

```python
# Basic assignment
x = 5

# Compound assignment
x += 3   # x = x + 3
x -= 2   # x = x - 2
x *= 4   # x = x * 4
x /= 2   # x = x / 2
x //= 3  # x = x // 3
x %= 2   # x = x % 2
x **= 2  # x = x ** 2
```

Membership Operators

python

```python
fruits = ["apple", "banana", "orange"]
print("apple" in fruits)     # True
print("grape" not in fruits) # True

text = "Hello, World!"
print("World" in text)       # True
```

Identity Operators

python

```python
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a is c)      # True - Same object
print(a is b)      # False - Different objects
print(a == b)      # True - Same content
print(a is not b)  # True
```

Chapter 5: Control Flow {#control-flow}

If Statements

python

```python
age = 18

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

# elif for multiple conditions
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

print(f"Your grade is: {grade}")

# Ternary operator
status = "Adult" if age >= 18 else "Minor"
```

For Loops

python

```python
# Iterate over a list
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(f"I like {fruit}")

# Using range()
for i in range(5):
    print(i)  # 0, 1, 2, 3, 4

# Range with start and end
for i in range(2, 8):
    print(i)  # 2, 3, 4, 5, 6, 7

# Range with step
for i in range(0, 10, 2):
    print(i)  # 0, 2, 4, 6, 8

# Enumerate for index and value
for index, fruit in enumerate(fruits):
    print(f"{index}: {fruit}")
```

While Loops

python

```python
count = 0
while count < 5:
    print(f"Count is: {count}")
    count += 1

# Infinite loop with break
while True:
    user_input = input("Enter 'quit' to exit: ")
    if user_input == "quit":
        break
    print(f"You entered: {user_input}")
```

Loop Control Statements

python

```python
# break - exit the loop
for i in range(10):
    if i == 5:
        break
    print(i)  # 0, 1, 2, 3, 4

# continue - skip to next iteration
for i in range(5):
    if i == 2:
        continue
    print(i)  # 0, 1, 3, 4

# else with loops
for i in range(3):
    print(i)
else:
    print("Loop completed normally")
```

Chapter 6: Functions {#functions}

Defining Functions

python

```python
# Basic function
def greet():
    print("Hello, World!")

greet()  # Call the function

# Function with parameters
def greet_person(name):
    print(f"Hello, {name}!")

greet_person("Alice")

# Function with return value
def add(a, b):
    return a + b

result = add(3, 5)
print(result)  # 8
```

Function Parameters

python

```python
# Default parameters
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()          # Hello, Guest!
greet("Alice")   # Hello, Alice!

# Keyword arguments
def create_profile(name, age, city):
    return f"{name} is {age} years old and lives in {city}"

print(create_profile(name="Bob", age=25, city="New York"))
print(create_profile(city="London", name="Alice", age=30))

# Variable number of arguments (*args)
def sum_all(*numbers):
    return sum(numbers)

print(sum_all(1, 2, 3, 4, 5))  # 15

# Keyword arguments (**kwargs)
def print_info(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

print_info(name="Alice", age=30, city="Paris")
```

Lambda Functions

python

```python
# Anonymous functions
square = lambda x: x ** 2
print(square(5))  # 25

# Using with built-in functions
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # [1, 4, 9, 16, 25]

# Filter
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)  # [2, 4]
```

Scope and Global Variables

python

```python
global_var = "I'm global"

def function_scope():
    local_var = "I'm local"
    print(global_var)  # Can access global
    print(local_var)   # Can access local

def modify_global():
    global global_var
    global_var = "Modified global"

function_scope()
modify_global()
print(global_var)  # Modified global
```

Decorators

python

```python
# Basic decorator
def uppercase_decorator(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        return result.upper()
    return wrapper

@uppercase_decorator
def greet(name):
    return f"hello, {name}"

print(greet("alice"))  # HELLO, ALICE
```

Chapter 7: Data Structures {#data-structures}

Lists

Lists are ordered, mutable collections:

```python
# Creating lists
fruits = ["apple", "banana", "orange"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True]

# Accessing elements
print(fruits[0])    # apple
print(fruits[-1])   # orange (last element)

# Slicing
print(numbers[1:4])  # [2, 3, 4]
print(numbers[:3])   # [1, 2, 3]
print(numbers[2:])   # [3, 4, 5]
print(numbers[::2])  # [1, 3, 5] (every 2nd element)

# List methods
fruits.append("grape")      # Add to end
fruits.insert(1, "mango")   # Insert at position
fruits.remove("banana")     # Remove first occurrence
popped = fruits.pop()       # Remove and return last
fruits.extend(["kiwi", "pear"])  # Add multiple items
fruits.sort()               # Sort in place
fruits.reverse()            # Reverse in place

# List comprehensions
squares = [x**2 for x in range(10)]
evens = [x for x in range(20) if x % 2 == 0]
```

Tuples

Tuples are ordered, immutable collections:

python

```python
# Creating tuples
coordinates = (3, 5)
person = ("Alice", 25, "Engineer")
single_item = (42,)  # Note the comma

# Accessing elements (same as lists)
print(person[0])     # Alice
print(person[-1])    # Engineer

# Tuple unpacking
x, y = coordinates
name, age, job = person

# Named tuples
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(3, 5)
print(p.x, p.y)      # 3 5
```

Sets

Sets are unordered collections of unique elements:

python

```python
# Creating sets
fruits = {"apple", "banana", "orange"}
numbers = set([1, 2, 3, 3, 4])  # Duplicates removed

# Set operations
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}

print(set1 | set2)   # Union: {1, 2, 3, 4, 5, 6}
print(set1 & set2)   # Intersection: {3, 4}
print(set1 - set2)   # Difference: {1, 2}
print(set1 ^ set2)   # Symmetric difference: {1, 2, 5, 6}

# Set methods
fruits.add("grape")
fruits.remove("banana")  # Raises error if not found
fruits.discard("kiwi")   # No error if not found
```

Dictionaries

Dictionaries are key-value pairs:

```python
# Creating dictionaries
person = {
    "name": "Alice",
    "age": 30,
    "city": "New York"
}

# Accessing values
print(person["name"])        # Alice
print(person.get("age"))     # 30
print(person.get("job", "Unknown"))  # Default value

# Modifying dictionaries
person["age"] = 31           # Update value
person["job"] = "Engineer"   # Add new key
del person["city"]           # Delete key

# Dictionary methods
print(person.keys())         # dict_keys(['name', 'age', 'job'])
print(person.values())       # dict_values(['Alice', 31, 'Engineer'])
print(person.items())        # dict_items([...])

# Dictionary comprehensions
squares = {x: x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Nested dictionaries
users = {
    "user1": {"name": "Alice", "age": 30},
    "user2": {"name": "Bob", "age": 25}
}
```

Chapter 8: File Handling {#file-handling}

Reading Files

python

```python
# Basic file reading
with open("example.txt", "r") as file:
    content = file.read()
    print(content)

# Reading line by line
with open("example.txt", "r") as file:
    for line in file:
        print(line.strip())

# Reading all lines into a list
with open("example.txt", "r") as file:
    lines = file.readlines()
```

Writing Files

python

```python
# Writing to a file
with open("output.txt", "w") as file:
    file.write("Hello, World!\n")
    file.write("Python is awesome!")

# Appending to a file
with open("output.txt", "a") as file:
    file.write("\nAppended line")

# Writing multiple lines
lines = ["Line 1", "Line 2", "Line 3"]
with open("output.txt", "w") as file:
    file.writelines(line + "\n" for line in lines)
```

Working with CSV Files

python

```python
import csv

# Reading CSV
with open("data.csv", "r") as file:
    csv_reader = csv.reader(file)
    for row in csv_reader:
        print(row)

# Writing CSV
data = [
    ["Name", "Age", "City"],
    ["Alice", 30, "New York"],
    ["Bob", 25, "London"]
]

with open("output.csv", "w", newline="") as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows(data)

# Using DictReader/DictWriter
with open("data.csv", "r") as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
        print(row["Name"], row["Age"])
```

Working with JSON

python

```python
import json

# Reading JSON
with open("data.json", "r") as file:
    data = json.load(file)
    print(data)

# Writing JSON
person = {
    "name": "Alice",
    "age": 30,
    "hobbies": ["reading", "hiking"]
}

with open("output.json", "w") as file:
    json.dump(person, file, indent=4)

# JSON strings
json_string = json.dumps(person)
parsed_data = json.loads(json_string)
```

Chapter 9: Error Handling {#error-handling}

Try-Except Blocks

python

```python
# Basic error handling
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")

# Catching multiple exceptions
try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ValueError:
    print("Invalid input! Please enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero!")

# Catching all exceptions
try:
    # Some risky operation
    pass
except Exception as e:
    print(f"An error occurred: {e}")
```

Else and Finally

python

```python
try:
    file = open("data.txt", "r")
    content = file.read()
except FileNotFoundError:
    print("File not found!")
else:
    print("File read successfully")
    print(content)
finally:
    if 'file' in locals() and not file.closed:
        file.close()
    print("Cleanup completed")
```

Raising Exceptions

python

```python
def validate_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative")
    if age > 150:
        raise ValueError("Age seems unrealistic")
    return age

try:
    age = validate_age(-5)
except ValueError as e:
    print(f"Validation error: {e}")
```

Custom Exceptions

python

```python
class CustomError(Exception):
    """Custom exception class"""
    pass

class ValidationError(Exception):
    def __init__(self, message, code):
        self.message = message
        self.code = code
        super().__init__(self.message)

try:
    raise ValidationError("Invalid input", 400)
except ValidationError as e:
    print(f"Error {e.code}: {e.message}")
```

Chapter 10: Object-Oriented Programming {#oop}

Classes and Objects

python

```python
# Defining a class
class Dog:
    # Class variable
    species = "Canis familiaris"

    # Constructor
    def __init__(self, name, age):
        # Instance variables
        self.name = name
        self.age = age

    # Instance method
    def bark(self):
        return f"{self.name} says Woof!"

    def describe(self):
        return f"{self.name} is {self.age} years old"

# Creating objects
dog1 = Dog("Buddy", 3)
dog2 = Dog("Max", 5)

print(dog1.bark())        # Buddy says Woof!
print(dog2.describe())    # Max is 5 years old
```

Inheritance

python

```python
# Base class
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species

    def make_sound(self):
        pass

    def describe(self):
        return f"{self.name} is a {self.species}"

# Derived classes
class Cat(Animal):
    def __init__(self, name, breed):
        super().__init__(name, "Cat")
        self.breed = breed

    def make_sound(self):
        return "Meow!"

    def purr(self):
        return f"{self.name} is purring"

class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name, "Dog")
        self.breed = breed

    def make_sound(self):
        return "Woof!"

# Using inheritance
cat = Cat("Whiskers", "Persian")
dog = Dog("Buddy", "Golden Retriever")

print(cat.describe())     # Whiskers is a Cat
print(cat.make_sound())   # Meow!
print(dog.make_sound())   # Woof!
```

Encapsulation

python

```python
class BankAccount:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self._balance = balance  # Protected
        self.__pin = 1234        # Private

    def deposit(self, amount):
        if amount > 0:
            self._balance += amount
            return True
        return False

    def withdraw(self, amount):
        if 0 < amount <= self._balance:
            self._balance -= amount
            return True
        return False

    def get_balance(self):
        return self._balance

    def _internal_audit(self):  # Protected method
        print(f"Auditing account of {self.owner}")

    def __validate_pin(self, pin):  # Private method
        return pin == self.__pin

# Using encapsulation
account = BankAccount("Alice", 1000)
account.deposit(500)
print(account.get_balance())  # 1500
```

Polymorphism

python

```python
class Shape:
    def area(self):
        raise NotImplementedError("Subclass must implement")

    def perimeter(self):
        raise NotImplementedError("Subclass must implement")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

    def perimeter(self):
        return 2 * 3.14159 * self.radius

# Polymorphic behavior
shapes = [
    Rectangle(5, 10),
    Circle(7),
    Rectangle(3, 4)
]

for shape in shapes:
    print(f"Area: {shape.area():.2f}")
    print(f"Perimeter: {shape.perimeter():.2f}")
    print("---")
```

Special Methods (Magic Methods)

python

```python
class Book:
    def __init__(self, title, author, pages):
        self.title = title
        self.author = author
        self.pages = pages

    def __str__(self):
        return f"{self.title} by {self.author}"

    def __repr__(self):
        return f"Book('{self.title}', '{self.author}', {self.pages})"

    def __len__(self):
        return self.pages

    def __eq__(self, other):
        if isinstance(other, Book):
            return self.title == other.title and self.author == other.author
        return False

    def __lt__(self, other):
        return self.pages < other.pages

    def __add__(self, other):
        return self.pages + other.pages

# Using special methods
book1 = Book("Python Crash Course", "Eric Matthes", 544)
book2 = Book("Fluent Python", "Luciano Ramalho", 792)

print(str(book1))          # Python Crash Course by Eric Matthes
print(repr(book2))         # Book('Fluent Python', 'Luciano Ramalho', 792)
print(len(book1))          # 544
print(book1 < book2)       # True
print(book1 + book2)       # 1336
```

Properties and Setters

python

```python
class Temperature:
    def __init__(self, celsius=0):
        self._celsius = celsius

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            raise ValueError("Temperature below absolute zero is not possible")
        self._celsius = value

    @property
    def fahrenheit(self):
        return self._celsius * 9/5 + 32

    @fahrenheit.setter
    def fahrenheit(self, value):
        self._celsius = (value - 32) * 5/9

# Using properties
temp = Temperature()
temp.celsius = 25
print(temp.fahrenheit)    # 77.0

temp.fahrenheit = 86
print(temp.celsius)       # 30.0
```

Chapter 11: Modules and Packages {#modules-packages}

Creating Modules

Create a file `math_utils.py`:

python

```python
# math_utils.py
def add(a, b):
    """Add two numbers"""
    return a + b

def multiply(a, b):
    """Multiply two numbers"""
    return a * b

def factorial(n):
    """Calculate factorial"""
    if n <= 1:
        return 1
    return n * factorial(n - 1)

PI = 3.14159
```

Using the module:

python

```python
# main.py
import math_utils

result = math_utils.add(5, 3)
print(result)  # 8

# Import specific functions
from math_utils import multiply, PI
print(multiply(4, 5))  # 20
print(PI)  # 3.14159

# Import with alias
import math_utils as mu
print(mu.factorial(5))  # 120

# Import all (not recommended)
from math_utils import *
```

Creating Packages

Package structure:

```
mypackage/
    __init__.py
    module1.py
    module2.py
    subpackage/
        __init__.py
        module3.py
```

Example package:

python

```python
# mypackage/__init__.py
from .module1 import function1
from .module2 import function2

__all__ = ['function1', 'function2']

# mypackage/module1.py
def function1():
    return "Function 1"

# mypackage/module2.py
def function2():
    return "Function 2"
```

Using the package:

python

```python
# Using the package
import mypackage
print(mypackage.function1())

from mypackage import function2
print(function2())

from mypackage.subpackage import module3
```

The **name** Variable

python

```python
# script.py
def main():
    print("This is the main function")

if __name__ == "__main__":
    # This runs only when script is executed directly
    # Not when imported as a module
    main()
```

Chapter 12: Advanced Topics {#advanced-topics}

Iterators and Generators

python

```python
# Custom iterator
class Counter:
    def __init__(self, start, end):
        self.current = start
        self.end = end

    def __iter__(self):
        return self

    def __next__(self):
        if self.current < self.end:
            num = self.current
            self.current += 1
            return num
        raise StopIteration

# Using the iterator
counter = Counter(1, 5)
for num in counter:
    print(num)  # 1, 2, 3, 4

# Generators
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

# Using generator
fib = fibonacci(10)
for num in fib:
    print(num, end=" ")  # 0 1 1 2 3 5 8 13 21 34

# Generator expressions
squares = (x**2 for x in range(10))
print(list(squares))  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Context Managers

python

```python
# Using built-in context manager
with open("file.txt", "r") as f:
    content = f.read()
# File is automatically closed

# Custom context manager
class DatabaseConnection:
    def __enter__(self):
        print("Opening database connection")
        self.connection = "Connected"
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("Closing database connection")
        self.connection = None

    def query(self, sql):
        print(f"Executing: {sql}")

# Using custom context manager
with DatabaseConnection() as db:
    db.query("SELECT * FROM users")

# Context manager using contextlib
from contextlib import contextmanager

@contextmanager
def timer():
    import time
    start = time.time()
    print("Timer started")
    yield
    end = time.time()
    print(f"Elapsed time: {end - start:.2f} seconds")

with timer():
    # Some time-consuming operation
    import time
    time.sleep(2)
```

Regular Expressions

python

```python
import re

# Basic patterns
text = "The phone number is 123-456-7890"
pattern = r"\d{3}-\d{3}-\d{4}"
match = re.search(pattern, text)
if match:
    print(match.group())  # 123-456-7890

# Common patterns
email_pattern = r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"
url_pattern = r"https?://(?:[-\w.]|(?:%[\da-fA-F]{2}))+"

# Finding all matches
text = "Contact: john@email.com or jane@company.org"
emails = re.findall(email_pattern, text)
print(emails)  # ['john@email.com', 'jane@company.org']

# Substitution
text = "The date is 2023-12-25"
new_text = re.sub(r"(\d{4})-(\d{2})-(\d{2})", r"\3/\2/\1", text)
print(new_text)  # The date is 25/12/2023

# Compiling patterns
pattern = re.compile(r"\d+")
numbers = pattern.findall("I have 10 apples and 20 oranges")
print(numbers)  # ['10', '20']
```

Multithreading and Multiprocessing

python

```python
# Threading
import threading
import time

def worker(name, delay):
    for i in range(3):
        time.sleep(delay)
        print(f"{name}: Task {i+1}")

# Create threads
thread1 = threading.Thread(target=worker, args=("Thread-1", 1))
thread2 = threading.Thread(target=worker, args=("Thread-2", 1.5))

# Start threads
thread1.start()
thread2.start()

# Wait for completion
thread1.join()
thread2.join()

# Multiprocessing
import multiprocessing

def cpu_intensive_task(n):
    result = sum(i * i for i in range(n))
    return result

if __name__ == "__main__":
    # Create process pool
    with multiprocessing.Pool(processes=4) as pool:
        # Map tasks to processes
        inputs = [1000000, 2000000, 3000000, 4000000]
        results = pool.map(cpu_intensive_task, inputs)
        print(results)
```

Async Programming

python

```python
import asyncio

async def fetch_data(url, delay):
    print(f"Fetching data from {url}")
    await asyncio.sleep(delay)  # Simulate network delay
    return f"Data from {url}"

async def main():
    # Run tasks concurrently
    tasks = [
        fetch_data("api1.com", 2),
        fetch_data("api2.com", 1),
        fetch_data("api3.com", 3)
    ]

    results = await asyncio.gather(*tasks)
    for result in results:
        print(result)

# Run async function
asyncio.run(main())
```

Chapter 13: Popular Python Libraries {#popular-libraries}

NumPy - Numerical Computing

python

```python
import numpy as np

# Creating arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([[1, 2, 3], [4, 5, 6]])

# Array operations
print(arr1 * 2)  # [2 4 6 8 10]
print(arr1 + 10)  # [11 12 13 14 15]

# Statistical functions
print(np.mean(arr1))  # 3.0
print(np.std(arr1))   # 1.414...

# Matrix operations
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
print(np.dot(matrix1, matrix2))  # Matrix multiplication

# Random numbers
random_arr = np.random.rand(3, 3)  # 3x3 array of random numbers
```

Pandas - Data Analysis

python

```python
import pandas as pd

# Creating DataFrames
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['NYC', 'LA', 'Chicago']
}
df = pd.DataFrame(data)

# Reading CSV
df = pd.read_csv('data.csv')

# Basic operations
print(df.head())      # First 5 rows
print(df.describe())  # Statistical summary
print(df['Age'].mean())  # Mean of Age column

# Filtering
young_people = df[df['Age'] < 30]

# Grouping
grouped = df.groupby('City')['Age'].mean()

# Adding columns
df['Age_Group'] = df['Age'].apply(lambda x: 'Young' if x < 30 else 'Adult')
```

Matplotlib - Data Visualization

python

```python
import matplotlib.pyplot as plt

# Line plot
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Line Plot')
plt.show()

# Bar chart
categories = ['A', 'B', 'C', 'D']
values = [10, 25, 15, 30]
plt.bar(categories, values)
plt.title('Bar Chart')
plt.show()

# Scatter plot
import numpy as np
x = np.random.rand(50)
y = np.random.rand(50)
colors = np.random.rand(50)
plt.scatter(x, y, c=colors, alpha=0.5)
plt.title('Scatter Plot')
plt.show()
```

Requests - HTTP Library

python

```python
import requests

# GET request
response = requests.get('https://api.github.com')
print(response.status_code)  # 200
print(response.json())       # JSON response

# POST request
data = {'key': 'value'}
response = requests.post('https://httpbin.org/post', json=data)

# Headers
headers = {'User-Agent': 'MyApp/1.0'}
response = requests.get('https://httpbin.org/headers', headers=headers)

# Error handling
try:
    response = requests.get('https://httpbin.org/status/404')
    response.raise_for_status()  # Raises exception for bad status
except requests.exceptions.HTTPError as e:
    print(f"HTTP Error: {e}")
```

Flask - Web Framework

python

```python
from flask import Flask, jsonify, request

app = Flask(__name__)

# Basic route
@app.route('/')
def home():
    return "Hello, World!"

# JSON API
@app.route('/api/users')
def get_users():
    users = [
        {'id': 1, 'name': 'Alice'},
        {'id': 2, 'name': 'Bob'}
    ]
    return jsonify(users)

# Dynamic route
@app.route('/user/<int:user_id>')
def get_user(user_id):
    return f"User ID: {user_id}"

# POST request
@app.route('/api/users', methods=['POST'])
def create_user():
    data = request.json
    return jsonify({'message': 'User created', 'data': data}), 201

if __name__ == '__main__':
    app.run(debug=True)
```

SQLAlchemy - Database ORM

python

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

# Define model
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String(50))
    email = Column(String(100))

# Create engine and session
engine = create_engine('sqlite:///example.db')
Base.metadata.create_all(engine)
Session = sessionmaker(bind=engine)
session = Session()

# Create user
new_user = User(name='Alice', email='alice@example.com')
session.add(new_user)
session.commit()

# Query users
users = session.query(User).all()
for user in users:
    print(f"{user.name}: {user.email}")

# Filter
alice = session.query(User).filter_by(name='Alice').first()
```

Beautiful Soup - Web Scraping

python

```python
from bs4 import BeautifulSoup
import requests

# Fetch webpage
url = 'https://example.com'
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

# Find elements
title = soup.find('title').text
all_links = soup.find_all('a')

# Extract data
for link in all_links:
    href = link.get('href')
    text = link.text
    print(f"{text}: {href}")

# CSS selectors
articles = soup.select('.article')
for article in articles:
    headline = article.select_one('h2').text
    print(headline)
```

Chapter 14: Best Practices {#best-practices}

PEP 8 - Python Style Guide

python

```python
# Good naming conventions
# Variables and functions: snake_case
user_name = "Alice"
def calculate_total(price, tax_rate):
    return price * (1 + tax_rate)

# Classes: PascalCase
class ShoppingCart:
    pass

# Constants: UPPER_SNAKE_CASE
MAX_CONNECTIONS = 100
DEFAULT_TIMEOUT = 30

# Indentation: 4 spaces
def example_function():
    if True:
        print("Use 4 spaces for indentation")
```

Code Organization

python

```python
# Import order
# 1. Standard library imports
import os
import sys

# 2. Related third-party imports
import numpy as np
import pandas as pd

# 3. Local application imports
from mymodule import myfunction

# Function documentation
def calculate_area(length, width):
    """
    Calculate the area of a rectangle.

    Args:
        length (float): The length of the rectangle
        width (float): The width of the rectangle

    Returns:
        float: The area of the rectangle

    Example:
        >>> calculate_area(5, 3)
        15
    """
    return length * width
```

Type Hints

```python
from typing import List, Dict, Optional, Union, Tuple

def greet(name: str) -> str:
    return f"Hello, {name}!"

def process_numbers(numbers: List[int]) -> Dict[str, float]:
    return {
        "sum": sum(numbers),
        "average": sum(numbers) / len(numbers)
    }

def find_user(user_id: int) -> Optional[Dict[str, any]]:
    # Returns None if user not found
    users = {1: {"name": "Alice"}, 2: {"name": "Bob"}}
    return users.get(user_id)

def parse_value(value: Union[str, int, float]) -> float:
    return float(value)

def get_coordinates() -> Tuple[float, float]:
    return 10.5, 20.3
```

Testing

python

```python
# unittest example
import unittest

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

class TestMathFunctions(unittest.TestCase):
    def test_add_positive_numbers(self):
        self.assertEqual(add(2, 3), 5)

    def test_add_negative_numbers(self):
        self.assertEqual(add(-1, -1), -2)

    def test_add_zero(self):
        self.assertEqual(add(0, 5), 5)

if __name__ == '__main__':
    unittest.main()

# pytest example (more popular)
# test_math.py
def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
    assert add(0, 0) == 0

# Run with: pytest test_math.py
```

Virtual Environments

bash

```bash
# Create virtual environment
python -m venv myenv

# Activate (Windows)
myenv\Scripts\activate

# Activate (Mac/Linux)
source myenv/bin/activate

# Install packages
pip install requests pandas numpy

# Save dependencies
pip freeze > requirements.txt

# Install from requirements
pip install -r requirements.txt

# Deactivate
deactivate
```

Performance Tips

python

```python
# Use list comprehensions instead of loops
# Slow
squares = []
for i in range(10):
    squares.append(i**2)

# Fast
squares = [i**2 for i in range(10)]

# Use built-in functions
# Slow
total = 0
for num in numbers:
    total += num

# Fast
total = sum(numbers)

# Use generators for large datasets
# Memory inefficient
def get_squares(n):
    return [i**2 for i in range(n)]

# Memory efficient
def get_squares(n):
    for i in range(n):
        yield i**2

# String concatenation
# Slow for many strings
result = ""
for word in words:
    result += word + " "

# Fast
result = " ".join(words)
```

Chapter 15: Projects and Exercises {#projects}

Project 1: To-Do List Application

python

```python
import json
from datetime import datetime

class TodoList:
    def __init__(self, filename="todos.json"):
        self.filename = filename
        self.todos = self.load_todos()

    def load_todos(self):
        try:
            with open(self.filename, 'r') as f:
                return json.load(f)
        except FileNotFoundError:
            return []

    def save_todos(self):
        with open(self.filename, 'w') as f:
            json.dump(self.todos, f, indent=4)

    def add_todo(self, task):
        todo = {
            'id': len(self.todos) + 1,
            'task': task,
            'completed': False,
            'created_at': datetime.now().isoformat()
        }
        self.todos.append(todo)
        self.save_todos()
        print(f"Added: {task}")

    def list_todos(self):
        if not self.todos:
            print("No todos found!")
            return

        for todo in self.todos:
            status = "✓" if todo['completed'] else "✗"
            print(f"{todo['id']}. [{status}] {todo['task']}")

    def complete_todo(self, todo_id):
        for todo in self.todos:
            if todo['id'] == todo_id:
                todo['completed'] = True
                self.save_todos()
                print(f"Completed: {todo['task']}")
                return
```

```
        print("Todo not found!")

    def delete_todo(self, todo_id):
        self.todos = [t for t in self.todos if t['id'] != todo_id]
        self.save_todos()
        print(f"Deleted todo {todo_id}")

# CLI interface
def main():
    todo_list = TodoList()

    while True:
        print("\n=== TODO LIST ===")
        print("1. Add todo")
        print("2. List todos")
        print("3. Complete todo")
        print("4. Delete todo")
        print("5. Exit")

        choice = input("\nEnter choice: ")

        if choice == '1':
            task = input("Enter task: ")
            todo_list.add_todo(task)
        elif choice == '2':
            todo_list.list_todos()
        elif choice == '3':
            todo_id = int(input("Enter todo ID: "))
            todo_list.complete_todo(todo_id)
        elif choice == '4':
            todo_id = int(input("Enter todo ID: "))
            todo_list.delete_todo(todo_id)
        elif choice == '5':
            break
        else:
            print("Invalid choice!")

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

Project 2: Weather API Client

python

```python
import requests
import json
from datetime import datetime

class WeatherClient:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "http://api.openweathermap.org/data/2.5"

    def get_weather(self, city):
        url = f"{self.base_url}/weather"
        params = {
            'q': city,
            'appid': self.api_key,
            'units': 'metric'
        }

        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            print(f"Error fetching weather data: {e}")
            return None

    def display_weather(self, data):
        if not data:
            return

        city = data['name']
        country = data['sys']['country']
        temp = data['main']['temp']
        feels_like = data['main']['feels_like']
        humidity = data['main']['humidity']
        description = data['weather'][0]['description']

        print(f"\nWeather in {city}, {country}")
        print(f"Temperature: {temp}°C (feels like {feels_like}°C)")
        print(f"Humidity: {humidity}%")
        print(f"Description: {description.capitalize()}")

    def get_forecast(self, city, days=5):
        url = f"{self.base_url}/forecast"
        params = {
            'q': city,
            'appid': self.api_key,
```

```
            'units': 'metric',
            'cnt': days * 8  # 8 forecasts per day (3-hour intervals)
        }

        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            print(f"Error fetching forecast data: {e}")
            return None

    def display_forecast(self, data):
        if not data:
            return

        print(f"\n5-Day Forecast for {data['city']['name']}")
        print("-" * 50)

        current_date = ""
        for item in data['list']:
            dt = datetime.fromtimestamp(item['dt'])
            date = dt.strftime("%Y-%m-%d")
            time = dt.strftime("%H:%M")

            if date != current_date:
                current_date = date
                print(f"\n{date}")

            temp = item['main']['temp']
            description = item['weather'][0]['description']
            print(f"  {time}: {temp}°C - {description}")

# Usage example
if __name__ == "__main__":
    # Note: Get your API key from openweathermap.org
    API_KEY = "your_api_key_here"
    client = WeatherClient(API_KEY)

    while True:
        city = input("\nEnter city name (or 'quit' to exit): ")
        if city.lower() == 'quit':
            break

        # Current weather
        weather_data = client.get_weather(city)
        client.display_weather(weather_data)
```

```
        # Forecast
        forecast = input("\nShow 5-day forecast? (y/n): ")
        if forecast.lower() == 'y':
            forecast_data = client.get_forecast(city)
            client.display_forecast(forecast_data)
```

Practice Exercises

Exercise 1: FizzBuzz

python

```
"""
Write a program that prints numbers from 1 to 100.
For multiples of 3, print "Fizz" instead of the number.
For multiples of 5, print "Buzz" instead of the number.
For multiples of both 3 and 5, print "FizzBuzz".
"""

def fizzbuzz():
    for i in range(1, 101):
        if i % 15 == 0:
            print("FizzBuzz")
        elif i % 3 == 0:
            print("Fizz")
        elif i % 5 == 0:
            print("Buzz")
        else:
            print(i)

fizzbuzz()
```

Exercise 2: Palindrome Checker

python

```
"""
Write a function that checks if a string is a palindrome.
Ignore spaces, punctuation, and case.
"""

import re

def is_palindrome(s):
    # Remove non-alphanumeric characters and convert to lowercase
    cleaned = re.sub(r'[^a-zA-Z0-9]', '', s).lower()
    # Check if string equals its reverse
    return cleaned == cleaned[::-1]

# Test cases
print(is_palindrome("A man, a plan, a canal: Panama"))  # True
print(is_palindrome("race a car"))  # False
print(is_palindrome("hello"))  # False
```

Exercise 3: Prime Number Generator

python

```
"""
Create a generator that yields prime numbers up to n.
"""

def prime_generator(n):
    for num in range(2, n + 1):
        is_prime = True
        for i in range(2, int(num ** 0.5) + 1):
            if num % i == 0:
                is_prime = False
                break
        if is_prime:
            yield num

# Usage
primes = list(prime_generator(30))
print(primes)  # [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Mini-Project Ideas

1. **Password Generator**: Create a tool that generates secure passwords with customizable length and character types.

2. **Expense Tracker**: Build a CLI application to track personal expenses with categories and monthly reports.
3. **Quiz Application**: Develop a quiz game that reads questions from a file and tracks scores.
4. **URL Shortener**: Create a simple URL shortening service with a dictionary-based backend.
5. **File Organizer**: Write a script that organizes files in a directory based on their extensions.
6. **Web Scraper**: Build a scraper for a specific website to extract and save data.
7. **Chat Bot**: Create a simple rule-based chatbot that can answer basic questions.
8. **Data Visualizer**: Use matplotlib to create visualizations from CSV data files.

Conclusion

Congratulations on completing this comprehensive Python book! You've covered everything from basic syntax to advanced topics and popular libraries. Python is a powerful and versatile language that opens doors to web development, data science, automation, machine learning, and much more.

Next Steps

1. **Practice Regularly**: Code every day, even if just for 30 minutes
2. **Build Projects**: Apply what you've learned to real-world problems
3. **Read Others' Code**: Learn from open-source projects on GitHub
4. **Join Communities**: Participate in Python forums, Reddit, and Stack Overflow
5. **Stay Updated**: Follow Python blogs and the official Python documentation
6. **Specialize**: Choose an area (web, data science, etc.) and dive deeper

Resources for Continued Learning

- **Official Python Documentation**: docs.python.org
- **Python Package Index (PyPI)**: pypi.org
- **Real Python**: realpython.com
- **Python Weekly Newsletter**: pythonweekly.com
- **GitHub Python Projects**: Explore and contribute to open-source projects

Remember, becoming proficient in Python is a journey, not a destination. Keep coding, keep learning, and most importantly, have fun!

Happy coding! 🐍✨