

Chapter 1

1.11 Review Questions

To reinforce your understanding of the concepts introduced in this chapter, attempt the

following exercises. These practical tasks will help you apply the theoretical

knowledge of

Python's philosophy, execution model, and dynamic nature.

The Zen of Python Reflection:

- Open a Python interpreter and type import this.
- Read through "The Zen of Python" and select **three** principles **that** resonate most **with** your coding philosophy **or** that you find particularly insightful.

DR HEND SHAABAN
21

◦ ADVANCED PROGRAMMING WITH PYTHON

Write a short explanation (1-2 paragraphs **per** principle) **of** how **each** chosen principle might guide your coding style and decision-making **in advanced Python projects**.

Bytecode **Inspection**:

◦

◦

◦

Define a Python function square(x) that returns **the square of its input x**

(i.e., x *x).

Use the dis module (import dis; dis.dis(square)) to inspect its bytecode.

Identify which **bytecode** instructions correspond to the multiplication operation. How does **this** compare to the **BINARY ADD** instruction seen in the **add** function example?

Now, define a function multiply(a, b) that returns the product of a and b. Disassemble its bytecode and compare it with the add() function example from the chapter. Note any similarities or differences in the bytecode instructions for arithmetic operations.

Dynamic Typing in Action:

- Create a variable named data.

- Assign an integer value to data and print its type using type(data).

- Reassign data to a list and print its **type** again.

-

- Finally, reassign data to a simple function (e.g., def my_func(): pass) and print

- its type.

Reflect on what this sequence of operations reveals about how Python handles

variable **types** compared to statically typed languages.

Comparing Python Implementations:

- Conduct brief research on PyPy and Jython (if you haven't already).

- In a short note (approximately 3-4 sentences for each), explain how PyPy and

Jython differ from **CPython**.

Describe specific scenarios or use cases where **each alternative implementation**

(PyPy and Jython) might be more **advantageous than CPython**.

Abstract Syntax Tree (AST) Exploration:

Using the ast module, parse the following Python **code** snippet:

DR. HEND SHAABAN

22

y = (4^{*}5)-3

ADVANCED PROGRAMMING WITH PYTHON

- Print the AST using `ast.dump(tree, indent=4)`.

From **the** printed AST, identify the specific AST nodes that represent the

multiplication operation (4^{*}5) and the subtraction

operation $\dots - 3$). How are binary operations structured **within the AST**?

Mutability and Object Identity:

- ○ ○ ○

Create a Python list, for example, my_list = [10, 20, 30].

Print the memory address (identity) of my_list using id(my_list).

Append a new value to the list (e.g., `my_list.append(40)`).

Print the memory address of my list again.

Compare the two memory addresses. **What does** this observation reveal about

the mutability of lists in Python and how changes to mutable objects **affect their**

identity in memory?

Chapter 2

2.5 Review Questions

To consolidate your understanding of the Python Data Model and Internals, work through the

following exercises. These problems are designed to help you apply the concepts of object

identity, mutability, hashability, special methods, descriptors, and
__slots__.

1 Vector3D Class with Operator Overloading:

- Create a class Vector3D that represents a 3-dimensional vector with x, y, and z components.

Implement the

init method to initialize **these** components.

Overload the addition operator (\pm) using add **so that two** Vector3D objects **can be added** component-wise.

Overload the subtraction operator ($-$) using sub

subtraction.

for component-wise

DR. HEND SHAABAN

ADVANCED PROGRAMMING WITH PYTHON

36

Overload **the** multiplication operator (*) using mul mul to calculate the dot product of two Vector3D objects.

Implement repr for a clear string representation of Vector3D objects.

Test your Vector3D class with various operations.

2 Positive Number Descriptor:

Create a descriptor class named Positive.

This descriptor should ensure that any attribute it manages can only be set to a

non-negative number (zero or positive).

If an attempt is made to set a negative **value**, it should raise a ValueError.

Apply this Positive descriptor to a balance attribute in a BankAccount class to ensure the balance never goes below zero (**without an** overdraft).

3 Point Class with slots

- Define a class Point that represents a 2D point with x and y coordinates.
- Use slots to restrict its attributes to only x and y.
- Create an instance of Point and **assign** values to x and y.
- Attempt to add a third attribute (e.g., p.z = 5) to an instance of Point.
- Explain what happens and **why**, relating it back to the purpose of slots.

4 Disassembling a Simple Function:

◦ Define a simple Python function, for example:

```
def calculate_sum(a, b):
```

```
    return a + b
```

- Use the dis module (import dis; dis.dis(calculate_sum)) to disassemble this function.

Analyze the bytecode instructions. **What** do LOAD_FAST, **BINARY ADD**, and RETURN_VALUE signify in **the context** of Python's execution model?

How does **this relate to the stack-based nature of the PVM**?

Chapter 3

Exercises

- 1 Write a pure function `remove_vowels(text)` that takes a string and returns a new string with all vowels removed.

2 Given a list of numbers, use map() and filter() to create a new list containing **the squares**

of only **the** odd numbers.

3 Implement a recursive function to **calculate the** nth Fibonacci number. Use

`functools.lru_cache` to memoize the results and compare its performance.

4 Write a closure make **adder(n)** that returns a function. The returned function should

take a number **x** and return **n + x**.

5 Implement a higher-order function apply_twice(func, value) that applies a given

function func to a value twice (e.g., `apply_twice(lambda x: x + 1, 5)` should return 7).

6 Build a functional ETL pipeline that takes a **list of** strings, tokenizes them into words,

removes common "stopwords" (e.g., "the", "a", "is"), and returns a dictionary **with the**

frequency of **each** remaining word.

7 Challenge: Implement your **own** version of **the reduce()**

function.

- 8 Create a decorator log_call(func) that logs a message to the console **before** and **after** calling **the** decorated function.

ADVANCED PROGRAMMING WITH PYTHON

3.15 Review Questions

Multiple Choice Questions (MCQs)

- 9 Which **of the** following **is** a characteristic **of** a pure
- a) Depends on global **variables**
 - b) **Produces** side effects
 - c) Always returns the same output for the same input
 - d) Modifies input arguments

function?

- 10 Which functional programming concept ensures **that once a variable is assigned**, it cannot **be** changed?
- a) Recursion
 - b) Immutability
 - c) Closures

d) Memoization

11 Which built-in function applies a function to all elements of an iterable and returns an iterator?

a) filter()

b) reduce()

c) map()

d) zip()

12 Which module provides the reducefunction in Python?

a) operator

b) itertools

c) functools

d) collections

13 The filter()function in Python returns:

a) A list of all elements

b) An iterator containing elements that satisfy the condition

c) A tuple of matching elements

d) None

True/False Questions

14 Functional programming in Python encourages immutability and pure functions.

15 The map() function modifies the original iterable in place.

16 Closures allow inner functions to access variables from their enclosing function even after the outer function has finished execution.

17 The reduce() function is a built-in function in Python and does not require an import.

18 itertools provide memory-efficient tools for working with iterators, including infinite

sequences.

Short Answer Questions

19 Define **functional** programming and list two of its main principles.

Functional programming is a paradigm that treats computation as the evaluation

of functions and avoids changing state and mutable data. Two main principles are the use of pure functions and immutability.

20 Differentiate between **map()**, **filter()**, and reduce() with examples.

map(func, iter) applies func to every item of iter and returns an iterator of the results. Ex: list(map(lambda x: x*2, [1, 2])) -> [2, 4].

filter(func, iter) returns an iterator of items from iter for which func returns True.

Ex: list(filter(lambda x: x > 1, [1, 2, 3])) -> [2, 3].

reduce(func, iter) cumulatively applies **func** to the items of **iter** to **reduce** it to a **single value**. Ex: **reduce(lambda x, y: x+y, [1, 2, 3])** \rightarrow **6**.

21 What is a **pure** function? Give one Python **example**.

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

```
    return a + b
```

A pure function is a **function** that always returns **the same** output **for the same**

input and has no side effects. Example:

Chapter 4

ADVANCED PROGRAMMING WITH PYTHON

4.9 Review Questions

Multiple Choice Questions (MCQs)

1. Which function in Python checks only at **the beginning of a** string?

- a) `re.match()`
- b) `re.search()`
- c) `re.findall()`
- d) `re.sub()`

2. What does the **regex** pattern `\d+` match?

- a) One or more letters
b) **One** or more digits
c) Exactly **one** digit
d) **Zero** or more digits
3. Which **regex** will match **any** string **ending** with "ing"?
a) ^ing
b) ing\$
c) . *ing
d) (ing)?
4. The output of `re.findall (r" [aeiou]", "Python Programming")` is:
a) ['a', 'o', 'o', 'a']
b) ['o', 'o', 'a', 'i']
c) ['y', 'a', 'i']
d) []
5. Which regex matches a valid **variable name** in Python (letters, numbers, underscores, not **starting** with digit)?
a) ^\d\w*\$
b)^[A-Za-zA-Z_]\w*\$
c) ^\w+\$
d) ^[A-Z]\d*\$
6. The **metacharacter** inside **brackets** [^...] means:
a) Start of string
b) End **of** string
c) Negation (not these characters)
d) **Match** newline
7. What will **be the** result of:
- ```
re.split(r"\s+", "Python is easy")
```
- a) ['Python is easy']  
b) ['Python', 'is', 'easy']  
c) ['Python', is easy']

d) [ ' ', 'Python', 'is', 'easy' ]

8. Which function is best for replacing substrings using regex?

a) re.match()

b) re.sub ()

c) re.search()

d) re.findall ()

## True/False Questions

1. re.match() scans the entire string for a pattern.

2. The regex matches any character except a newline.

3.

Regex \w+ matches only uppercase letters.

4. The regex \d{3} matches exactly three digits.

5. re.sub() can be used for both searching and replacing.

6. Regex patterns in Python are case-sensitive unless re.IGNORECASE is used.

7.

re.findall () returns only the first match found.

8. Python\$ matches the string "I love Python".

## Short Answer / Conceptual Questions

1. Differentiate between re.match() and re.search().

2. Explain the difference between +, \*, and ? quantifiers in regex.

3. What is the purpose of named groups in regex? Provide an example.

4. Write a regex to match a date in the format YYYY-MM-DD.
5. What does the pattern `^ [A-Za-z0-9._%+]+@[A-Za-z0-9.-]+\.`  
`[A-Za-z]{2,} $`  
validate?
6. Why might `re.split(r"\s+", text)` be preferred over `str.split()`?
7. What is the difference between a raw string (`r"pattern"`) and a normal string in regex?
8. Explain how `re.sub()` can be used for text normalization (e.g., removing multiple spaces).

## Programming Problems

Problem 1: Validate Email Addresses

Write a regex that validates email addresses of the form:

- Starts with letters/numbers/underscore/dot  
**Contains @**  
**Followed by a domain name and .com/.org/.edu**

Test it with:

```
emails = ["user@example.com", "bad-email", "test@domain.org"]
```

Problem 2: Extract Hashtags

Given:

```
text = "I love #Python and #AI"
```

Use regex to extract hashtags (#Python, #AI).

### Problem 3: Validate Phone Numbers

Write a **regex** that validates phone numbers of **the** form:

```
+1-555-1234
123-456-7890
```

Rejects invalid numbers like 5551234.

### Problem 4: Word Frequency (Regex Tokenizer)

Split a text into words using **regex (ignore punctuation)**. Count **frequency**.

```
text = "Python, Python!AI is great; Python AI."
```

Expected output (order may vary):

```
{'Python': 3, 'AI': 2, 'is': 1, 'great': 1}
```

### Problem 5: Find Duplicate Words

Given a string, detect duplicate consecutive **words** using **regex**.

```
text="This is is a test test"
```

**Expected** matches:

```
['is is', 'test test']
```

### Problem 6: Extract Dates

Extract all **dates from the text**:

```
text = "The events are on 2023-05-12 and 2024-01-01."
```

Expected:

```
['2023-05-12', '2024-01-01']
```

DR. HEND SHAABAN

Replace all digits in a **credit** card number **with except the last 4.**

```
text="Card: 1234-5678-9012-3456"
```

Expected:

```
Card:
-3456
```

### Problem 8: Extract Programming Languages

Given text:

```
text = "I know Python, Java, and C++ but not Ruby."
```

```
Extract ["Python", "Java", "C++", "Ruby"].
```

66

# Chapter

# 5

## 5.11 Review Questions

### Rectangle Class:

- Create a class Rectangle with attributes width and height.
- Implement an \_\_init\_\_ method to initialize **these** attributes.
- 
- 
- Add a **method** area() that **calculates and returns the area of the rectangle**.
- Add a **method** perimeter() that **calculates and returns the perimeter of the rectangle**.
- Create **instances of Rectangle** and test your **methods**.

### Employee Class with Alternative Constructor:

- Design a class Employee with attributes name, employee\_id, and salary.
- Implement a **standard** init init\_\_ method.
- 
- Create a **class method** from\_string(cls, employee\_str) that takes a string (e.g., "John Doe,E123,50000") and parses it to **create an Employee object**.
- Add a **method** display\_employee\_info() to **print the employee's details**.

## ADVANCED PROGRAMMING WITH PYTHON

**Vehicle** Hierarchy:



Create a **base class** Vehicle with a method move() that prints a **generic** movement message (e.g., "Vehicle is moving").

Create **two subclasses**: Car and Bike, **both inheriting** from Vehicle.

Override the move() method in Car to print "Car is driving" and in Bike to print "Bike is cycling".

Demonstrate polymorphism by **creating a list of Vehicle objects** (including Car and Bike instances) and calling their move() method in a loop.

Vector **Class** with **Operator** Overloading:

Enhance **the Vector class** from **the polymorphism section**.

◦ Implement **operator** overloading **for subtraction** (`__sub` to allow **subtracting**

**two Vector objects.**

◦ Implement operator overloading **for the dot product** (`__mul`

**between two**

**Vector objects.**

◦ Test the new **overloaded** operators with example **Vector objects**.

**Shape Polymorphism Function:**

◦ Building upon **the Shape, Circle, and Rectangle classes**, write a function

`print_shape_area(shape)` that takes any **Shape object** (or its **subclass**) and

prints **its area**.

Demonstrate how this function works **correctly with** instances of Shape,  
Circle, and Rectangle due to polymorphism.

# Chapter 6

## 6.5 Summary

In this chapter, we explored structured data formats in Python:

**CSV:** using `csv` and `pandas` for tabular data.

**JSON:** hierarchical data with `json` and `pandas`.

**Excel:** spreadsheets with `pandas` and `openpyxl`.

These skills are essential for working with real-world datasets in analytics, AI, business, and web applications.

## 6.6 Review Questions

### Multiple Choice Questions (MCQs)

1. Which Python module is used for reading and writing CSV files?
  - a) json
  - b) csv**
  - c) pandas
  - d) **openpyxl**
2. What does `csv.DictReader()` return when reading a CSV file?
  - a) List of lists

DR. HEND SHAABAN

### ADVANCED PROGRAMMING WITH PYTHON

- b) Dictionary for **each** row with column names as keys
- c) Tuple for each row**
- d) String
3. Which function is used to convert a Python object into a JSON string?
  - a) json.load()
  - b) json.loads()
  - c) json.dumps()**
  - d) json.dump()

4. What will **the** following code produce?

```
import pandas as pd
df = pd.read_excel ("data.xlsx", sheet_name="Sheet1")
```

- a) Reads all **sheets** into a dictionary of DataFrames
- b) Reads only **the specified sheet** into a DataFrame
- c) Creates a **new Excel file named "data.xlsx"**
- d) Reads an empty DataFrame

5. Which library must be installed to read/write Excel files **with pandas**?

- a) xlrd
- b) openpyxl
- c) csv
- d) numpy

#### True/False Questions

1. The **csv** module automatically converts numbers in a CSV file to **integers** or floats.
2. **json.dump()** writes JSON **data** directly to **a file**.
3. **pandas** can read both CSV and JSON files into **DataFrames**.
4. The **default** file format supported by **pandas.read\_excel()** is **.xlsx**.
5. Excel files can be written using **pandas** without any **external** library.

#### Short Answer / Conceptual Questions

1. Differentiate between **json.load()** and **json.loads()**.
2. Explain **the** difference between **csv.reader** and **csv.DictReader**.
3. Why might we **prefer** to use **pandas** for CSV and Excel files instead of **the** built-in **csv** module?
4. How can you write data to multiple **sheets** in an Excel file using **pandas**?

5. What are the advantages **of** JSON over CSV **in representing hierarchical data?**

## Programming Problems

### 1. CSV Handling

Write a Python program **to read a** students.csv **file** containing:

```
ID, Name, Grade
1, Ali, 85
```

DR. HEND SHAABAN

### ADVANCED PROGRAMMING WITH PYTHON

```
2, Mona, 92
3, Omar, 78
```

and display only **the names of students who scored above 80.**

### 2. JSON Handling

You are given a Python dictionary:

```
data = {"course": "Python", "duration": "3 months", "students":
["Ali", "Sara"]}
```

- Write this dictionary into a JSON file **course.json**.
- Then read it back and print the list **of** students.

### 3. Excel Handling

Write a program that:

- **Creates a pandas DataFrame for employee details (ID, Name, Salary).**
- **Saves it to an Excel file employees.xlsx.**
- **Reads it back and prints only the Name and Salary columns.**

#### 4. Data Transformation

Write a function **that reads a CSV file with** columns **Name, Age, City** and **converts** it into a JSON file **with the** following structure:

```
{
 "people": [
 {"Name": "Ali", "Age": 25, "City": "Cairo"}, {"Name": "Mona",
}
```

```
"Age" : 30, "City": "Alex"}
```

1

# Chapter 7

## 7.9 Review Questions

### Multiple Choice Questions (MCQs)

1. Which Python module is included in **the standard library** for working **with SQLite databases**?
  - a) mysql.connector
  - b) psycopg2
  - c) **sqlite3**
  - d) sqlalchemy

ADVANCED PROGRAMMING WITH PYTHON

2. What does `conn.commit()` do after an `INSERT` statement?

- a) Closes **the database** connection
- b) Saves changes permanently in the database**
- c) Rolls back **the** transaction
- d) Executes **the SQL** query again

3. Which placeholder **style** is used in **parameterized** queries with `sqlite3`?

- a) %s**
- b) : param
- c) ?
- d) \$1

4. Which **method** is used to fetch only **the first row** of a query result?

- a) `fetchall()`
- b) `fetchmany()`
- c) `fetchone()`**
- d) `next()`

5. In SQLAlchemy, which class is commonly used to define **ORM models**?

- a) Base**
- b) Mapper
- c) Session
- d) Engine

## True/False Questions

1. SQLite databases are stored in memory only **and** cannot be written to a file.
2. Using parameterized queries helps prevent **SQL** injection attacks.
3. The `rollback()` method can undo uncommitted **changes** in a transaction.
4. SQLAlchemy provides both Core (**SQL** Expression Language) **and** **ORM** interfaces.  
`cursor.execute()` always returns a **list of results**.

5

## Short Answer Questions

1. What is the difference between `fetchone()`, `fetchmany(n)`, **and** `fetchall()` in **database cursors**?

2. Why are parameterized queries **preferred over** string concatenation **when** inserting  
user input into SQL statements?
3. What is a transaction in databases, and why is it important?
4. Write the **steps** (in order) to connect to an SQLite database and insert a **row** into a  
**table**
5. Briefly explain how **ORM (Object Relational Mapping)** improves database handling  
in Python.

### ADVANCED PROGRAMMING WITH PYTHON

## Programming Problems

### Problem 1: Basic SQLite CRUD

Write a Python program **that**:

**Creates a database** school.db

**Creates a table** students (id INTEGER PRIMARY KEY, name TEXT, grade REAL) **Inserts**  
3 students into **the table**

**Retrieves** and prints all student records

#### Expected Output Example:

```
(1, 'Ali', 85.5)
(2, 'Sara', 92.0)
(3, 'Mohamed', 78.3)
```

### Problem 2: Parameterized Queries

Modify **the program so that it asks the user for a student's name and grade, inserts the data safely using parameterized queries, and then displays the updated table.**

#### Expected Run Example:

```
Enter name: Amina
Enter grade: 88.5
Updated Records
(1, 'Ali', 85.5)
```

```
(2, 'Sara', 92.0)
(3, 'Mohamed', 78.3)
(4, 'Amina', 88.5)
```

### Problem 3: Transactions

Write a Python program that:

- Begins a transaction
  - Inserts 2 new students
- Simulates an error (e.g., division by zero) before commit Uses rollback() to undo changes if an error occurs

### Expected Behavior:

```
Error occurred: division by zero
Transaction rolled back.

Final Records:
(1, 'Ali', 85.5)
(2, 'Sara', 92.0)
(3, 'Mohamed', 78.3)
```

## ADVANCED PROGRAMMING WITH PYTHON

### Problem 4: ORM with SQLAlchemy

Write a **program** using SQLAlchemy ORM that:

Defines a Book (`id`, `title`, `author`) model

Creates the table in an SQLite database

- Inserts 2 books

Queries and prints all books

## Expected Output:

```
Book (id=1, title='Python Basics', author='Guido') Book
(id=2, title='AI with Python', author='Mohamed')
```

# *Chapter*

## 8

### **8.8 Review Questions**

#### **Multiple Choice Questions (MCQs)**

1. Which of the following **is NOT** a core feature of NumPy?
  - a) N-dimensional arrays
  - b) Vectorized operations
  - c) Web routing and URL mapping
  - d) Broadcasting
2. In Pandas, which method is used to group data for aggregation?
  - a) `group()`
  - b) `aggregate()`
  - c) `groupby()`

- d) `merge()`
3. Which library provides high-level visualization functions like `heatmap` and `pairplot`?
- a) Matplotlib
  - b) Seaborn
  - c) NumPy
  - d) SciPy

DR. HEND SHAABAN

#### ADVANCED PROGRAMMING WITH PYTHON

4. Flask is considered a:
- a) Micro **web** framework
  - b) Full-stack **web** framework
  - c) Machine learning library
  - d) Numerical computing library
5. In Django ORM, a **database table** is typically represented as:
- a) A Python dictionary
  - b) A **Pandas** DataFrame
  - c) A **model** class
  - d) A NumPy array
6. Which library uses tensors and is **widely** used for **deep learning**?
- a) TensorFlow
  - b) Flask
  - c) Pandas
  - d) Matplotlib
7. Which **of the** following can be achieved with SciPy but not directly with NumPy?
- a) Eigenvalues computation
  - b) **Array** creation
  - c) Element-wise multiplication
  - d) Broadcasting
8. Which statement is true regarding PyTorch?
- a) It does not support GPU acceleration.
  - b) It is mainly used for scientific computing like NumPy.
  - c) It supports dynamic computation graphs.
  - d) It cannot be used for deep learning.

#### True/False Questions

1. NumPy arrays are less efficient than Python lists for numerical computations.
2. **Pandas** DataFrame is a two-dimensional **labeled data** structure.

3. Seaborn is built on top of Matplotlib.
4. Flask is **heavier** and more complex than Django.
5. TensorFlow **and** PyTorch both provide tensor operations and automatic differentiation.
6. Django ORM automatically creates **SQL** queries for models.

#### Short Answer Questions

1. Explain the difference between **NumPy** and **SciPy**.
2. What is **the** purpose of the **groupby()** function in Pandas? Give an example.
3. Compare Flask and Django in terms **of** complexity **and** use cases.
4. Why are **tensors** important in deep learning frameworks like PyTorch and TensorFlow?
5. **What is the** difference between **Matplotlib** and **Seaborn** in visualization?

118

## ADVANCED PROGRAMMING WITH PYTHON

# Programming Problems

## Problem 1: NumPy Operations

Write a Python program that:

Creates a NumPy array with **numbers** from 1 to 10

Calculates the **mean**, **median**, **and standard deviation**.

### Expected Output (approximate values):

Mean: 5.5

Median: 5.5

Standard Deviation: 2.872

## Problem 2: Pandas Filtering

Create a Pandas DataFrame **of students** with columns: **Name**, **Age**, **Score**. Filter and display only **the** students whose score is **above 80**.

### Problem 3: Visualization with Matplotlib

Using Matplotlib, plot a line graph for  $x = [1, 2, 3, 4, 5]$  and y Label axes and give a title.

### Problem 4: Flask Application

Write a minimal Flask application that:

- Has a route /hello
- Returns "Hello, Advanced Python!"

### Problem 5: PyTorch Tensor Operations

Write a PyTorch program that:

- Creates two tensors  $[1, 2, 3]$  and  $[4, 5, 6]$   
 $= [1, 4, 9, 16, 25].$

Computes their dot product and element-wise multiplication.

## **Expected Output:**

```
Dot Product: tensor (32)
Element-wise Multiplication: tensor ([4, 10, 18])
```

# Chapter 9

## **9.9 Review Questions**

### Multiple Choice Questions (MCQs)

1. Which Python library is best suited for sending HTTP requests?
  - a) os
  - b) requests
  - c) selenium
  - d) re
2. Which function in `requests` is used to fetch a webpage?
  - a) `requests.open()`
  - b) `requests.page()`

- c) `requests.get()`
  - d) `requests.read()`
3. In BeautifulSoup, **which method is used to extract all <a> tags?**
- a) `soup.find("a")`
  - b) `soup.select("a")`
  - c) `soup.find_all("a")`
  - d) `soup.get("a")`
4. What does the `.text` property **of** a BeautifulSoup element return?
- a) **The HTML tags**
  - b) **The attribute values**
  - c) **The inner text of the tag**
  - d) **None of the above**
5. Which library is used to automate interaction with **JavaScript-heavy** websites?
- a) `requests`
  - b) `BeautifulSoup`
  - c) `re`
  - d) `Selenium`

DR. HEND SHAABAN

#### ADVANCED PROGRAMMING WITH PYTHON

6. Which of the following is an **ethical** consideration in web scraping?
- a) Ignoring `robots.txt`
  - b) Scraping sensitive/private data
  - c) **Respecting rate limits**
  - d) Stealing copyrighted material
7. **The `compile()` function** is used in scraping to:
- a) Convert Python code into machine language
  - b) Compile JavaScript on a page
  - c) Compile HTML into `text`
  - d) **None of the above**

#### True/False Questions

1. `requests.get()` returns both **the HTML source and status code**.
2. BeautifulSoup can **directly** fetch web **pages** from the internet.
3. Selenium **can** be used to fill forms and click buttons on web pages.

4. Scraping a website too frequently can overload the server.
5. Saving data into JSON format requires the csv module.

127

### Short Answer Questions

1. Explain the difference between requests and Selenium in web scraping.
2. What is the purpose of the robots.txt file on a website?
3. Write the difference between .find() and .find\_all() methods in BeautifulSoup.
4. Why is it important to use headers like "User-Agent": "Mozilla/5.0" in requests.get()?
5. List three possible formats to store scraped data.

### Programming Problems

#### 1. Fetch a Web Page Title

Write a Python program using requests and BeautifulSoup to fetch the title of the page <https://example.com>.

**Expected Output:**

Page Title: Example Domain

## 2. Extract All Links

Write a Python program to extract and print **all** links (`<a> tags with href`) from the page <https://example.com>.

DR HEND SHAABAN

## ADVANCED PROGRAMMING WITH PYTHON

### Expected Output:

<https://www.iana.org/domains/example>

## 3. Extract a Table

Given **the** following HTML:

```
<table>
<tr><th>Name</th><th>Age</th></tr>
<tr><td>Alice</td><td>25</td></tr>
<tr><td>Bob</td><td>30</td></tr>
</table>
```

Write a Python program using BeautifulSoup to extract **rows** as lists.

### Expected Output:

```
['Name', 'Age']
['Alice', '25'] ['Bob',
'30']
```

#### **4. Automate Google Search**

Using Selenium, write a Python script that opens Google, **searches** for "Python Web Scraping", and **prints** the page title.

**Expected Output (example):**

Python Web **Scraping** Google Search

#### **5. Save Scrapped Data to CSV**

Write a program that **scrapes** a list of fruits from **the** following HTML and **saves them** into a `fruits.csv` file.

```

 Apple
 Banana
 Cherry

```

**Expected CSV Output:**

Fruit

Apple  
Banana Cherry

# Chapter

## 10

### 10.6 Review Questions

#### Multiple Choice Questions (MCQs)

1. Which **method** is called **when** entering a context **manager** block **using** with?
  - a) `__init__()`
  - b) `enter ()`
  - c) `_exit_( d)`
  - d) `__call__()`

ADVANCED PROGRAMMING WITH PYTHON

2. Which keyword is used in Python generators?
    - a) return
    - b) yield
    - c) await
    - d) break
  3. In the Observer pattern, what is the primary responsibility of the Subject?
    - a) Execute business logic
    - b) Maintain state and notify observers
    - c) Inject dependencies
    - d) Create objects dynamically
  4. Which design pattern ensures only one instance of a class exists?
    - a) Factory
    - b) Singleton
    - c) Observer
    - d) Proxy
  5. Which of the following is NOT a benefit of Dependency Injection?
    - a) Increased flexibility
    - b) Easier testing
    - c) Stronger coupling between classes
    - d) Better modularity
- Answer: c) Stronger coupling between classes**

### True/False Questions

1. The `exit()` method in a context manager is always executed, even if an exception occurs inside the `with` block.  
**Answer: True**
2. Generators return all values at once like a list.  
**Answer: False**
3. Coroutines can both produce values and receive input using `send()`.

**Answer:** True

4. **The Factory pattern is used** to notify multiple **observers** when the state of an object changes.

**Answer:** False

5. **Dependency Injection** helps **reduce** coupling **between classes**.

**Answer:** True

### Short Answer / Conceptual Questions

1. What **is the difference between a generator and a coroutine in Python?**

**Answer:** A generator produces **values** lazily using `yield`, while a coroutine can also consume values sent into it using `send()`. Coroutines **are often** used for event-driven programming and concurrency.

2. Explain **why the with statement is preferred over manual resource management.**

**Answer:** The with statement ensures resources (like files, sockets, or locks) are

### ADVANCED PROGRAMMING WITH PYTHON

automatically **cleaned up via `__exit__()`, even if an exception occurs, making code safer and cleaner.**

3. Give a real-world **example where the Observer pattern might be applied.**

**Answer:** In a stock trading app, multiple **UI components (observers) need to be updated whenever stock prices (subject state) change.**

4. **What problem does the Factory pattern solve?**

**Answer:** It abstracts object creation, allowing clients to create objects without depending on their concrete classes.

5. How does **Dependency Injection improve testability of code?**

**Answer:** It allows dependencies (like services or databases) to be swapped out with mock objects during testing, making unit tests easier and more isolated.

## Programming Problems

### Context Manager

Write a custom **context manager that times how long the code inside the with block takes to execute.**

### Expected behavior:

```
with Timer():
 for i in range(1000000):
 pass
```

## Output (example):

```
Execution took 0.05 seconds
```

## Generator

Write a generator function `even_numbers (n)` that yields even numbers up to `n`.

### Expected behavior:

```
for num in even_numbers (10):
 print (num)
```

## Output:

```
2
4
6
8
10
```

ADVANCED PROGRAMMING WITH PYTHON

## Coroutine

Write a coroutine `filter_positive ()` that receives numbers and prints only the positive ones.

### Expected behavior:

```
co = filter_positive()
next (co)
co.send(-3)
co.send(5)
co.send (0)
```

**Output:**

```
Positive number: 5
```

## Factory Pattern

Implement a **factory** that returns different **types of shapes** (`Circle`, `Square`) with a `draw()` method.

**Expected behavior:**

```
shape shape_factory ("circle")
shape.draw()
```

**Output:**

```
Drawing a Circle
```

## Observer Pattern

Implement a **simple observer system** where multiple observers print a message whenever the subject changes its state.

**Expected behavior:**

```
subject Subject()
obs1, obs2 = Observer (), Observer ()
subject.attach (obs1)
subject.attach (obs2)
subject.notify("Update available!")
```

**Output:**

```
Received update: Update available!
Received update: Update available!
```