Chapter 1

الاسم : رنا اشرف محمد سعيد(2)

(1)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

◦ 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

solutions

1) The Zen of Python Reflection

Principle 1: “Simple is better than complex.”

This principle embodies one of the fundamental ideas behind Python’s design philosophy. In advanced programming, simplicity is not merely a stylistic preference, but a cornerstone for building scalable and maintainable systems. As software grows in size and complexity, unnecessary complications become a major barrier to readability, collaboration, and debugging. Adopting this principle helps developers focus on constructing clear, straightforward solutions that are easy to reason about and modify. In long-term projects, simplicity directly contributes to fewer bugs, easier optimization, and a cleaner architecture. Therefore, this principle guides advanced developers to resist clever, overly intricate solutions in favor of designs that are intuitive and transparent.

Principle 2: “Readability counts.”

Readability is central to Python and is one of the primary reasons the language is widely used in large-scale, collaborative projects. In advanced development environments, readable code is a necessity because it ensures that the logic and structure of a

program can be understood by other developers—or even by yourself months later. This principle encourages the use of clear naming, consistent formatting, and explicit logic, all of which make the internal workings of a program accessible. Readability significantly reduces the cognitive load required to maintain and extend the codebase. In short, readable code is more robust, easier to test, and more adaptable to future enhancements.

Principle 3: “There should be one— and preferably only one —obvious way to do it.”

This principle emphasizes consistency and uniformity in how solutions are implemented. When a language offers a single, obvious approach to solving a problem, developers can collaborate more efficiently because there is less ambiguity in how features should be written.
In advanced Python projects, adhering to this principle leads to codebases that are coherent and predictable. It prevents fragmentation in coding style and reduces the learning curve for new contributors. This principle ultimately reinforces Python’s goal of being explicit and clear, ensuring that the intent behind a piece of code remains unambiguous.

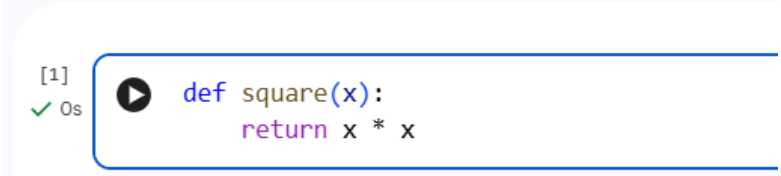
---

(2)Bytecode Inspection:

◦ Define a Python function square(x) that returns the square of its input x (i.e., x

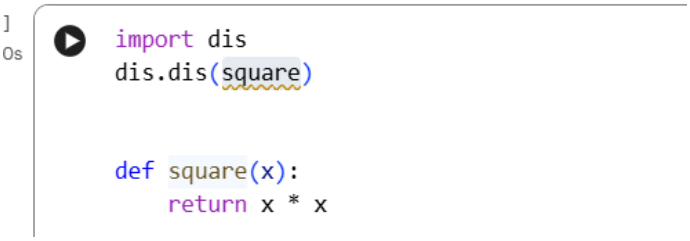
*** x)**

Solutions



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

Solutions



◦ **Identify which bytecode instructions correspond to the multiplication operation.**

How does this compare to the BINARY_ADD instruction seen in the add function example?

Solutions

```
import dis
dis.dis(square)


def square(x):
    return x * x
```

```
  1           0 RESUME                   0

  2           2 LOAD_FAST                0 (x)
              4 LOAD_FAST                0 (x)
              6 BINARY_OP                5 (*)
             10 RETURN_VALUE
```

Explanation:

The instruction **BINARY_MULTIPLY** corresponds to the multiplication operation. This is the multiply-equivalent of the **BINARY_ADD** instruction shown in the chapter's add() example. Both instructions operate on values loaded onto the Python Virtual Machine's stack, performing the required arithmetic, then pushing the result back onto the stack.

◦ **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.

Solutions

```
def multiply(a, b):
    return a * b
```

Comparison with add(a, b):

The only structural difference between multiply() and the add() example is the arithmetic opcode used:

- BINARY_ADD → addition
- BINARY_MULTIPLY → multiplication

Both follow the same pattern of loading operands, performing an operation, and returning the result. This highlights Python's uniform approach to arithmetic bytecode.

---

(3)Dynamic Typing in Action:

◦ Create a variable named data.

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

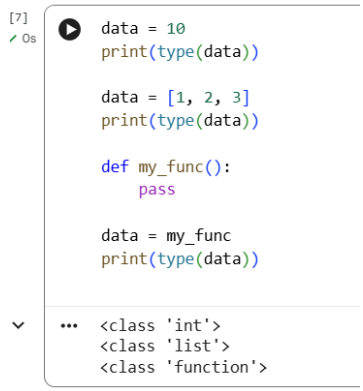
◦ Reassign data to a list and print its type again.

◦ 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

Solutions



```
[7]
0s
data = 10
print(type(data))

data = [1, 2, 3]
print(type(data))

def my_func():
    pass

data = my_func
print(type(data))
```

```
<class 'int'>
<class 'list'>
<class 'function'>
```

Reflection:

This sequence demonstrates Python's dynamic typing model, where variables do not have fixed types but are instead bound to objects whose types are determined at runtime. The variable data transitions from an integer to a list and then to a function reference without any type declarations or restrictions.
In contrast to statically typed languages such as Java or C++, where variable types must be defined ahead of time and cannot be changed, Python's flexibility enables rapid development and experimentation. However, this dynamic behavior also means that type-related errors may only surface at runtime, making thorough testing essential in advanced Python programming.

Abstract Syntax Tree (AST) Exploration:

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

◦ 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 (... - 3). How are

binary operations structured within the AST?

Solutions

```
import ast
source = "y = (4 * 5) - 3"
tree = ast.parse(source, mode='exec')
print(ast.dump(tree, indent=4))
```

```
Module(
    body=[
        Assign(
            targets=[
                Name(id='y', ctx=Store())],
            value=BinOp(
                left=BinOp(
                    left=Constant(value=4),
                    op=Mult(),
                    right=Constant(value=5)),
                op=Sub(),
                right=Constant(value=3)))],
    type_ignores=[])
```

Explanation:

Python represents binary operations using BinOp nodes, each of which contains three major components:

- left: the left operand
- op: the operator object (e.g., Mult(), Sub())
- right: the right operand

This hierarchical structure reflects Python's parsing of expressions into an abstract, machine-independent format before compilation into bytecode.

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?

Solutions

```
my_list = [10, 20, 30]
print(id(my_list))

my_list.append(40)
print(id(my_list))
```

```
135273208845696
135273208845696
```

Reflection:

Both printed memory addresses (object identities) remain identical. This behavior illustrates that lists in Python are **mutable** objects: their contents may change, but their identity remains the same.
Modifying a mutable object affects its internal state without reallocating a new object in memory. This contrasts with immutable types such as integers or strings, where any change creates an entirely new object with a new identity

Chapter 2

الاسم: رنا اشرف محمد سعيد(2)

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 (+) using __add__ so that two Vector3D objects

can be added component-wise.

◦ Overload the subtraction operator (-) using __sub__ for component-wise

Subtraction

◦ Overload the multiplication operator (*) using __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

Solutions

```python
class Vector3D:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def __add__(self, other):
        if isinstance(other, Vector3D):
            return Vector3D(self.x + other.x,
                            self.y + other.y,
                            self.z + other.z)
        return NotImplemented

    def __sub__(self, other):
        if isinstance(other, Vector3D):
            return Vector3D(self.x - other.x,
                            self.y - other.y,
                            self.z - other.z)
        return NotImplemented

    def __mul__(self, other):
        if isinstance(other, Vector3D):
            return (self.x * other.x +
                    self.y * other.y +
                    self.z * other.z)
        return NotImplemented

    def __repr__(self):
        return f"Vector3D({self.x}, {self.y}, {self.z})"


# Testing
v1 = Vector3D(1, 2, 3)
v2 = Vector3D(4, 5, 6)

print(v1 + v2)
print(v1 - v2)
print(v1 * v2)
```

```
Vector3D(5, 7, 9)
Vector3D(-3, -3, -3)
32
```

---

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).

Solutions

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

    def __get__(self, instance, owner):
        if instance is None:
            return self
        return instance.__dict__[self.name]

    def __set__(self, instance, value):
        if value < 0:
            raise ValueError(f"{self.name} must be non-negative")
        instance.__dict__[self.name] = value

    def __delete__(self, instance):
        del instance.__dict__[self.name]


class BankAccount:
    balance = Positive("balance")

    def __init__(self, balance):
        self.balance = balance
```

---

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__.

Solutions

```python
class Point:
    __slots__ = ("x", "y")

    def __init__(self, x, y):
        self.x = x
        self.y = y


p = Point(2, 3)
p.x = 10
p.y = 20


# p.z = 5
```

---

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?

Solutions

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

Chapter 3

الاسم:رنا اشرف محمد سعيد(2)

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

Solutions

```
def remove_vowels(text):
    vowels = "aeiouAEIOU"
    result = ""
    for ch in text:
        if ch not in vowels:
            result += ch
    return result

print(remove_vowels("Functional Programming in Python"))
```

```
Fnctnl Prgrmmng n Pythn
```

---

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

of only the odd numbers

Solutions

```
[30]
Os
numbers = [1, 2, 3, 4, 5, 6, 7]

odd_numbers = filter(lambda x: x % 2 == 1, numbers)

squared_odds = map(lambda x: x * x, odd_numbers)

result = list(squared_odds)

print(result)
```

```
[1, 9, 25, 49]
```

---

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

functools.lru_cache to memoize the results and compare its performance

Solutions

```python
from functools import lru_cache

@lru_cache(maxsize=None)
def fib(n):
    if n < 0:
        raise ValueError("n must be non-negative")
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)


print(fib(10))
print(fib(20))
```

```
55
6765
```

---

4 Write a closure make_adder(n) that returns a function. The returned function should

take a number x and return n + x.

Solutions

```python
def make_adder(n):
    def inner(x):
        return n + x
    return inner


add5 = make_adder(5)
add10 = make_adder(10)

print(add5(3))
print(add10(3))
```

```
8
13
```

---

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).

Solutions

```
13
```

```python
def apply_twice(func, value):
    return func(func(value))


result1 = apply_twice(lambda x: x + 1, 5)
print(result1)
result2 = apply_twice(lambda x: x * 2, 3)
print(result2)
```

```
7
12
```

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.

Solutions

[4]
✓ 0s

```python
from functools import reduce

def build_word_frequency(lines):
    stopwords = {"the", "a", "is", "in", "on", "and", "to"}


    normalized_lines = map(lambda s: s.strip().lower(), lines)

    tokenized = map(lambda s: s.split(), normalized_lines)

    all_words = []
    for words in tokenized:
        all_words.extend(words)


    filtered_words = filter(lambda w: w not in stopwords, all_words)


    def update_freq(freq_dict, word):
        if word in freq_dict:
            freq_dict[word] += 1
        else:
            freq_dict[word] = 1
        return freq_dict

    freq = reduce(update_freq, filtered_words, {})

    return freq

lines = [
    "The sky is blue",
    "Python is in the sky",
    "Blue is a color in Python"
]

freq_dict = build_word_frequency(lines)
print(freq_dict)
```

```
{'sky': 2, 'blue': 2, 'python': 2, 'color': 1}
```

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

Solutions

```
[5]
✓ 0s
def my_reduce(func, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        try:
            accumulator = next(it)
        except StopIteration:
            raise TypeError("my_reduce() of empty sequence with no initial value")
    else:
        accumulator = initializer

    for item in it:
        accumulator = func(accumulator, item)
    return accumulator


numbers = [1, 2, 3, 4]

total = my_reduce(lambda a, b: a + b, numbers)
print(total)

product = my_reduce(lambda a, b: a * b, numbers, 1)
print(product)
```

```
10
24
```

---

8 Create a decorator log_call(func) that logs a message to the console before and after

calling the decorated function.

Solutions

```
5]
0s
def log_call(func):
    def wrapper(*args, **kwargs):
        print(f"[LOG] About to call {func.__name__} with {args}, {kwargs}")
        result = func(*args, **kwargs)
        print(f"[LOG] Finished calling {func.__name__}, returned {result}")
        return result
    return wrapper


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


@log_call
def greet(name):
    return f"Hello, {name}!"


print(add(3, 4))
print(greet("Kareem"))
```

```
[LOG] About to call add with (3, 4), {}
[LOG] Finished calling add, returned 7
7
[LOG] About to call greet with ('Kareem',), {}
[LOG] Finished calling greet, returned Hello, Kareem!
Hello, Kareem!
```

3.15 Review Questions

Multiple Choice Questions (MCQs)

9 Which of the following is a characteristic of a pure function?

a) Depends on global variables

b) Produces side effects

c)Always returns the same output for the same input

d) Modifies input arguments

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

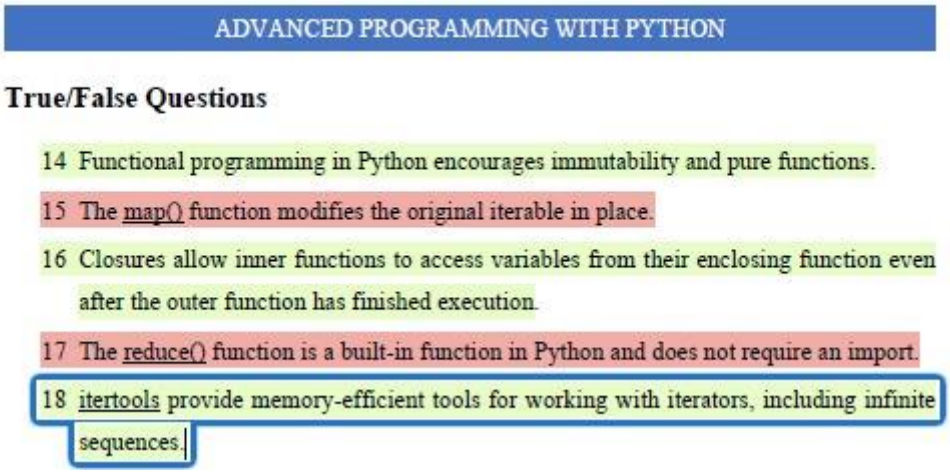
13 The filter()function in Python returns:

a) A list of all elements

b)An iterator containing elements that satisfy the condition

c) A tuple of matching elements

d) None



ADVANCED PROGRAMMING WITH PYTHON

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.

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.

Solutions

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing state or mutable data. Two main principles of functional programming are 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]) -> 6.

Solutions

- map(func, iterable):
  Applies a function to each element of an iterable and returns an iterator of the results.

```
s  list(map(lambda x: x * 2, [1, 2]))

...  [2, 4]
```

---

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

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

input and has no side effects. Example:

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

Solutions

A pure function is a function that always returns the same output for the same input and produces no side effects, such as modifying global variables or performing I/O operations.

```
[13]
✓ 0s  def add(a, b):
          return a + b
```

Chapter 4

الاسم:رنا اشرف محمد سعيد(2)

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-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"`.

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

Solutions

 re.match(pattern, text)
Tries to match the pattern **only at the beginning** of the string.

 re.search(pattern, text)
Searches the **entire string** and returns the first location where the pattern occurs.

---

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

Solutions

* → **0 or more** repetitions

- a* matches "", "a", "aa", "aaa", ...

+ → **1 or more** repetitions

- a+ matches "a", "aa", "aaa", but **not** ""

? → **0 or 1** repetition (optional)

- a? matches "" or "a"

---

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

Purpose:
Named groups make parts of a match easier to access by **name** instead of index, improving readability and maintainability.

---

4. Write a regex to match a date in the format YYYY-MM-DD

Solutions

\d{4} → 4 digits (year)
\d{2} → 2 digits (month)
\d{2} → 2 digits (day)

---

5. What does the pattern ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$

validate?

Solutions

It validates a **basic email address format**:

- Starts with one or more allowed characters for the **local part**: letters, digits, ., _, %, +, -
- Then @
- Then one or more letters/digits/dots/hyphens for the **domain**
- Then a dot .
- Then a top-level domain with at least **2 letters** (e.g., com, org, io, ...)

It's a common **email pattern**, not a perfect RFC validator but good for typical validation.

---

6. Why might re.split(r"\s+", text) be preferred over str.split()?

Solutions

re.split(r"\s+", text):

- Splits on **one or more whitespace characters**: spaces, tabs, newlines, etc.
- Treats multiple spaces, tabs, and mixed whitespace as **one separator**.
- Uses a **regex pattern**, so it's more flexible.

str.split():

- Without arguments: splits on any whitespace too, but you **cannot** express complex patterns (e.g., punctuation + spaces).

---

7. What is the difference between a raw string (`r"pattern"`) and a normal string in regex?

Solutions

- Normal string:
  Backslashes are processed by Python first.
  - Example: "\\d" actually represents a single backslash + d.
- Raw string (r"..."):
  Backslashes are not treated as escape characters by Python.
  - Example: r"\d" is passed to the regex engine as \d directly.

  Using raw strings avoids needing to double-escape patterns like \\d, \\b, \\s, etc., making regex more readable.

---

8. Explain how re.sub() can be used for text normalization (e.g., removing multiple spaces).

Solutions

[2]

```
import re

text = "This   is   a    test."
normalized = re.sub(r"\s+", " ", text)
print(normalized)
```

```
This is a test.
```

Chapter 5

الاسم:رنا اشرف محمد سعيد(2)

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

Solutions

[3] ✓ 0s

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

    def area(self):
        """Return the area of the rectangle."""
        return self.width * self.height

    def perimeter(self):
        """Return the perimeter of the rectangle."""
        return 2 * (self.width + self.height)

r1 = Rectangle(4, 5)
r2 = Rectangle(10, 2)

print(f"Rectangle 1: width={r1.width}, height={r1.height}")
print(f"Area: {r1.area()}")
print(f"Perimeter: {r1.perimeter()}")

print(f"Rectangle 2: width={r2.width}, height={r2.height}")
print(f"Area: {r2.area()}")
print(f"Perimeter: {r2.perimeter()}")
```

```
Rectangle 1: width=4, height=5
Area: 20
Perimeter: 18
Rectangle 2: width=10, height=2
Area: 20
Perimeter: 24
```

Employee Class with Alternative Constructor:

◦ Design a class Employee with attributes name, employee_id, and salary.

◦ Implement a standard __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.

Solutions

[4] 0s

```
class Employee:
    def __init__(self, name, employee_id, salary):
        self.name = name
        self.employee_id = employee_id
        self.salary = salary

    @classmethod
    def from_string(cls, employee_str):
        """Create an Employee instance from a comma-separated string."""
        name, employee_id, salary_str = employee_str.split(",")
        salary = float(salary_str)
        return cls(name, employee_id, salary)

    def display_employee_info(self):
        print(f"Name: {self.name}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Salary: {self.salary}")

emp1 = Employee("John Doe", "E123", 50000)
emp2 = Employee.from_string("Jane Smith,E456,60000")

emp1.display_employee_info()
print("-----")
emp2.display_employee_info()
```

```
Name: John Doe
Employee ID: E123
Salary: 50000
-----
Name: Jane Smith
Employee ID: E456
Salary: 60000.0
```

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.

Solutions

```
Salary: 60000.0
```

```
class Vehicle:
    def move(self):
        print("Vehicle is moving")


class Car(Vehicle):
    def move(self):
        print("Car is driving")


class Bike(Vehicle):
    def move(self):
        print("Bike is cycling")


vehicles = [
    Vehicle(),
    Car(),
    Bike()
]

for v in vehicles:
    v.move()
```

```
Vehicle is moving
Car is driving
Bike is cycling
```

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.

Solutions

[6] ✓ 0s

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __sub__(self, other):
        """Subtract two vectors component-wise."""
        if isinstance(other, Vector):
            return Vector(self.x - other.x, self.y - other.y)
        return NotImplemented

    def __mul__(self, other):
        """Compute the dot product of two vectors."""
        if isinstance(other, Vector):
            return self.x * other.x + self.y * other.y
        return NotImplemented

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"
v1 = Vector(2, 3)
v2 = Vector(5, 1)

v_sub = v1 - v2
v_dot = v1 * v2

print(f"v1: {v1}")
print(f"v2: {v2}")
print(f"v1 - v2 = {v_sub}")
print(f"v1 · v2 = {v_dot}")
```

```
v1: Vector(2, 3)
v2: Vector(5, 1)
v1 - v2 = Vector(-3, 2)
v1 · v2 = 13
```

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.

Solutions

[7]
0s

```python
import math

class Shape:
    def area(self):
        return 0
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * (self.radius ** 2)


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

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


def print_shape_area(shape):
    """Print the area of any Shape (or subclass of Shape)."""
    print(f"Area: {shape.area()}")

s = Shape()
c = Circle(3)
r = Rectangle(4, 5)

print_shape_area(s)
print_shape_area(c)
print_shape_area(r)
```

```
Area: 0
Area: 28.274333882308138
Area: 20
```

Chapter 6

الاسم:رنا اشرف محمد سعيد(2)

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

   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

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.

---

1. Differentiate between json.load() and json.loads().

Solutions

- `json.load()` reads JSON data **from a file object** and converts it into a Python object.
- `json.loads()` reads JSON data **from a string** and converts it into a Python object.

---

2. Explain the difference between csv.reader and csv.DictReader.

Solutions

- `csv.reader` reads a CSV file and returns each row as a **list of values**, accessed by index.
- `csv.DictReader` reads a CSV file and returns each row as a **dictionary**, where the keys are the column names and the values are the corresponding row data.

---

3. Why might we prefer to use pandas for CSV and Excel files instead of the built-in csv module

Solutions

- pandas provides higher-level abstractions such as DataFrames, supports powerful data manipulation, filtering, aggregation, and handles large datasets more efficiently. It also offers built-in support for reading and writing CSV, Excel, and JSON files with minimal code compared to the `csv` module.

4.How can you write data to multiple sheets in an Excel file using pandas?

Solutions

By using pandas.ExcelWriter, multiple DataFrames can be written to different sheets within the same Excel file.

---

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

Solutions

JSON supports nested and hierarchical structures, allowing data to be represented using objects and arrays. This makes it more suitable for complex data relationships. In contrast, CSV represents data in a flat tabular format, which makes it difficult to represent nested or hierarchical data accurately.

Chapter 7

الاسم:رنا اشرف محمد سعيد(2)

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`

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.
5. `cursor.execute()` always returns a list of results.

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

Solutions

fetchone() retrieves **one single row** from the result set.

fetchmany(n) retrieves the **next n rows** from the result set.

fetchall() retrieves **all remaining rows** from the result set at once.

**2. Why are parameterized queries preferred over string concatenation when inserting
user input into SQL statements?**

Solutions

Parameterized queries are preferred because they protect against SQL injection attacks by separating SQL code from user input. They also improve security, correctness, and allow the database engine to safely handle input values.

---

3. What is a transaction in databases, and why is it important?

Solutions

are executed as a single unit of work. It is important because it ensures data consistency and integrity by allowing changes to be either fully committed or fully rolled back in case of an error.

**4. Write the steps (in order) to connect to an SQLite database and insert a row into a
table.**

Solutions

Import the sqlite3 module.

Establish a connection using sqlite3.connect().

Create a cursor object using conn.cursor().

Execute an INSERT SQL statement using cursor.execute().

Commit the transaction using conn.commit().

Close the connection using conn.close().

---

5. Briefly explain how ORM (Object Relational Mapping) improves database handling in Python.

Solutions

developers to interact with the database using Python objects instead of writing raw SQL queries. It improves code readability, maintainability, and portability, and reduces the risk of SQL-related **errors.**

Chapter 8

الاسم:رنا اشرف محمد سعيد(2)

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

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.

---

1. Explain the difference between NumPy and SciPy.

Solutions

provides support for n-dimensional arrays, basic linear algebra, and fast element-wise operations. SciPy is built on top of NumPy and provides more advanced scientific and mathematical functionality such as optimization, statistics, signal processing, and numerical integration.

---

2. What is the purpose of the groupby() function in Pandas? Give an example.

Solutions

The groupby() function is used to split data into groups based on one or more columns and then apply aggregation functions such as sum, mean, or count.

---

3. Compare Flask and Django in terms of complexity and use cases.

Solutions

Flask is a lightweight micro web framework that is simple and flexible, making it suitable for small applications and REST APIs. Django is a full-stack web framework with many built-in features such as authentication, ORM, and admin panel, making it more complex but well suited for large and enterprise-level applications.

---

4. Why are tensors important in deep learning frameworks like PyTorch and TensorFlow?

Solutions

model parameters, and outputs. They enable efficient numerical computation, GPU acceleration, and automatic differentiation, which are essential for training deep learning models.

5.What is the difference between Matplotlib and Seaborn in visualization?

Solutions

Matplotlib is a low-l evel visualization library that provides detailed control over plots. Seaborn is built on top of Matplotlib and offers higher-level, more visually attractive statistical plots with simpler syntax.

]

Chapter 9

الاسم:رنا اشرف محمد سعيد(2)

1. Explain the difference between requests and Selenium in web scraping.

Solutions

requests is used to send HTTP requests and retrieve static HTML content quickly and efficiently. Selenium is a browser automation tool that can interact with web pages, execute JavaScript, fill forms, and click buttons, making it suitable for scraping dynamic, JavaScript-heavy websites.

2. What is the purpose of the robots.txt file on a website?

Solutions

The robots.txt file is used to inform web crawlers which parts of a website are allowed or disallowed for scraping. It helps control crawler behavior and encourages ethical web scraping.

Write the difference between .find() and .find_all() methods in BeautifulSoup. .4

Solutions

.find() returns the first matching element or None if no match is found.

.find_all() returns a list of all matching elements.

---

4. Why is it important to use headers like "User-Agent": "Mozilla/5.0" in requests.get()?

Solutions

Using a User-Agent header makes the request appear as if it is coming from a real browser. This helps avoid request blocking, access restrictions, and increases the chances of receiving the correct webpage content.

---

5. List three possible formats to store scraped data.

Solutions

CSV

JSON

Excel (XLSX)

Chapter 10

الاسم:رنا اشرف محمد سعيد(2)

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

Solutions

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.

Solutions

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

Solutions

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?

Solutions

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?

Solutions

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

FLASK TASK

الاسم:رنا اشرف محمد سعيد(2)

app.py

app.py > signup

```python

@app.before_request
def before_request():
    init_db()

@app.route('/')
def index():
    if "user_id" in session:
        return redirect(url_for('dashboard'))
    return redirect(url_for('login'))

@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username', '').strip()
        email = request.form.get('email', '').strip()
        password = request.form.get('password', '')
        confirm = request.form.get('confirm', '')

        if not username or not email or not password:
            flash('Please fill in all fields.')
            return redirect(url_for('signup'))

        if password != confirm:
            flash('Passwords do not match.')
            return redirect(url_for('signup'))

        hashed = generate_password_hash(password)
        db = get_db()
        try:
            db.execute(
                "INSERT INTO users (username, email, password) VALUES (?, ?, ?)",
                (username, email, hashed)
            )
            db.commit()
            flash('Account created successfully. You can now login.')
            return redirect(url_for('login'))
        except sqlite3.IntegrityError:
            flash('Username or email already exists.')
            return redirect(url_for('signup'))

    return render_template('signup.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
```

app.py > signup

```python
from flask import Flask, render_template, request, redirect, url_for, session, flash, g
import sqlite3
from werkzeug.security import generate_password_hash, check_password_hash
from functools import wraps
import os

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
DB_PATH = os.path.join(BASE_DIR, "users.db")

app = Flask(__name__)
app.secret_key = "change-this-to-a-random-secret"

def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = sqlite3.connect(DB_PATH)
        db.row_factory = sqlite3.Row
    return db

@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()

def init_db():
    db = get_db()
    db.execute("""
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            email TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL
        );
    """)
    db.commit()

def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if "user_id" not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function

```

app.py

app.py > signup

```python
def login():
    if request.method == 'POST':
        identifier = request.form.get('identifier', '').strip()
        password = request.form.get('password', '')

        db = get_db()
        user = db.execute(
            "SELECT * FROM users WHERE username = ? OR email = ?",
            (identifier, identifier)
        ).fetchone()

        if user and check_password_hash(user['password'], password):
            session.clear()
            session['user_id'] = user['id']
            session['username'] = user['username']
            flash('Logged in successfully.')
            return redirect(url_for('dashboard'))

        flash('Invalid username/email or password.')
        return redirect(url_for('login'))

    return render_template('login.html')

@app.route('/dashboard')
@login_required
def dashboard():
    username = session.get('username')
    return render_template('dashboard.html', username=username)

@app.route('/logout')
def logout():
    session.clear()
    flash('You have been logged out.')
    return redirect(url_for('login'))

if __name__ == '__main__':
    if not os.path.exists(DB_PATH):
        with sqlite3.connect(DB_PATH) as conn:
            conn.execute("""
                CREATE TABLE users (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT UNIQUE NOT NULL,
                    email TEXT UNIQUE NOT NULL,
                    password TEXT NOT NULL
                );
            """)
```

Assignment 1

الاسم:رنا اشرف محمد سعيد(2)

1. Transform and Clean Data

You are given a list of product names that contain extra spaces and inconsistent capitalization:

products = [" LAPTOP ", "phone ", " Tablet", "CAMERA "]

Using map() and lambda, clean the data by:

• Removing extra spaces

• Converting to title case (e.g. "laptop" → "Laptop")

Expected Output:

['Laptop', 'Phone', 'Tablet', 'Camera']

Hint:

Use str.strip() and str.title() inside a lamda

Solutions

```
products = [" LAPTOP ", "phone ", " Tablet", "CAMERA "]

clean_products = list(map(lambda p: p.strip().title(), products))
print(clean_products)
```

```
['Laptop', 'Phone', 'Tablet', 'Camera']
```

---

2. Convert Temperatures (Celsius → Fahrenheit) Given a list of temperatures in Celsius, convert them to Fahrenheit using: F=9/5C+32 celsius = [0, 10, 20, 30, 40] Expected Output: [32.0, 50.0, 68.0, 86.0, 104.0] Hint: Use map(lambda c: (9/5)*c + 32, Celsius)

Solutions

```
celsius = [0, 10, 20, 30, 40]

fahrenheit = list(map(lambda c: (9/5)*c + 32, celsius))
print(fahrenheit)
```

```
[32.0, 50.0, 68.0, 86.0, 104.0]
```

---

3. Apply Multiple Transformations You have a list of integers. You need to: 1. Square each number. 2. Then add 10 to each result. All using one map() and one lambda. nums = [1, 2, 3, 4, 5] Expected Output: [11, 14, 19, 26, 35] Hint: Combine both operations in one lambda.

Solutions

```
nums = [1, 2, 3, 4, 5]

result = list(map(lambda x: x**2 + 10, nums))
print(result)
```

```
[11, 14, 19, 26, 35]
```

---

4. Extract First and Last Characters Given a list of words, create a new list of tuples (first_char, last_char) for each word. words = ["python", "lambda", "programming", "map", "function"] Expected Output: [('p', 'n'), ('l', 'a'), ('p', 'g'), ('m', 'p'), ('f', 'n')] Hint: Use string indexing in the lambda: lambda w: (w[0], w[-1])

Solutions

```
words = ["python", "lambda", "programming", "map", "function"]

first_last = list(map(lambda w: (w[0], w[-1]), words))
print(first_last)
```

```
[('p', 'n'), ('l', 'a'), ('p', 'g'), ('m', 'p'), ('f', 'n')]
```

---

5. Nested Map Transformation (Challenge You have a 2D list representing student marks:) marks = [[45, 80, 70], [90, 60, 100], [88, 76, 92]] Using nested map() and lambda, • Increase each mark by 5%, • Round it to the nearest integer. Expected Output: [[47, 84, 74], [95, 63, 105], [92, 80, 97]] Hint: Use: map(lambda row: list(map(lambda x: round(x * 1.05), row)), marks).

Solutions

```
marks = [[45, 80, 70], [90, 60, 100], [88, 76, 92]]

new_marks = list(
    map(
        lambda row: list(map(lambda x: round(x * 1.05), row)),
        marks
    )
)

print(new_marks)
```

```
[[47, 84, 74], [94, 63, 105], [92, 80, 97]]
```

---

6. Create a program that normalizes a list of numbers between 0 and 1 using map() and lambda.

Solutions

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

min_val = min(numbers)
max_val = max(numbers)

normalized = list(map(lambda x: (x - min_val) / (max_val - min_val),
                      numbers))

print(normalized)
```

```
[0.0, 0.25, 0.5, 0.75, 1.0]
```

---

7. Given a list of sentences, extract the length of each word in every sentence using nested map().

Solutions

```
sentences = [
    "I love Python",
    "map and lambda are cool"
]

lengths = list(
    map(
        lambda s: list(map(lambda w: len(w), s.split())),
        sentences
    )
)

print(lengths)
```

```
[[1, 4, 6], [3, 3, 6, 3, 4]]
```

Web Scrapinga 1

الاسم:رنا اشرف محمد سعيد(2)

app.py | amazon.py 2

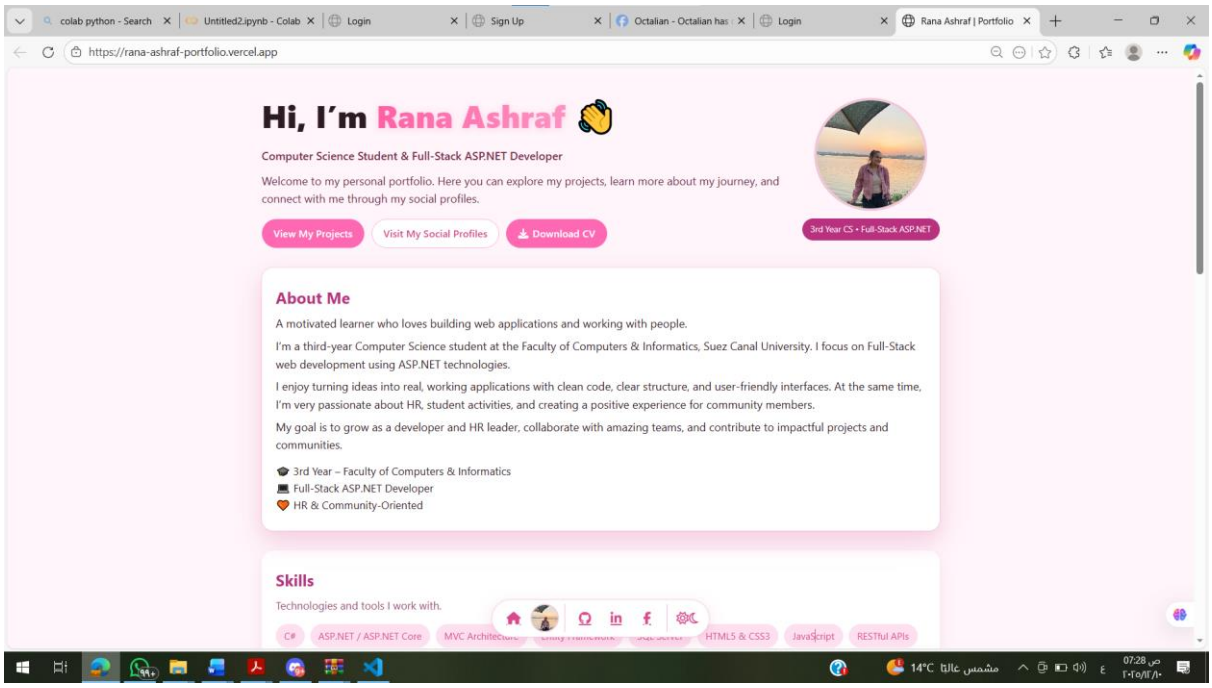
```python
import requests
from bs4 import BeautifulSoup

PRODUCT_URL = "https://www.amazon.com/dp/B07FZ8S74R"
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
    "Accept-Language": "en-US,en;q=0.9"
}

TARGET_PRICE = 900

response = requests.get(PRODUCT_URL, headers=HEADERS)
soup = BeautifulSoup(response.text, "html.parser")

title = soup.find(id="productTitle").get_text(strip=True)
price_tag = soup.find("span", class_="a-offscreen")
price_text = price_tag.get_text(strip=True)

price = float(price_text.replace("$", "").replace(",", ""))

print("Product:", title)
print("Current price:", price)

if price < TARGET_PRICE:
    print("less than goal")
else:
    print("greater than goal")
```

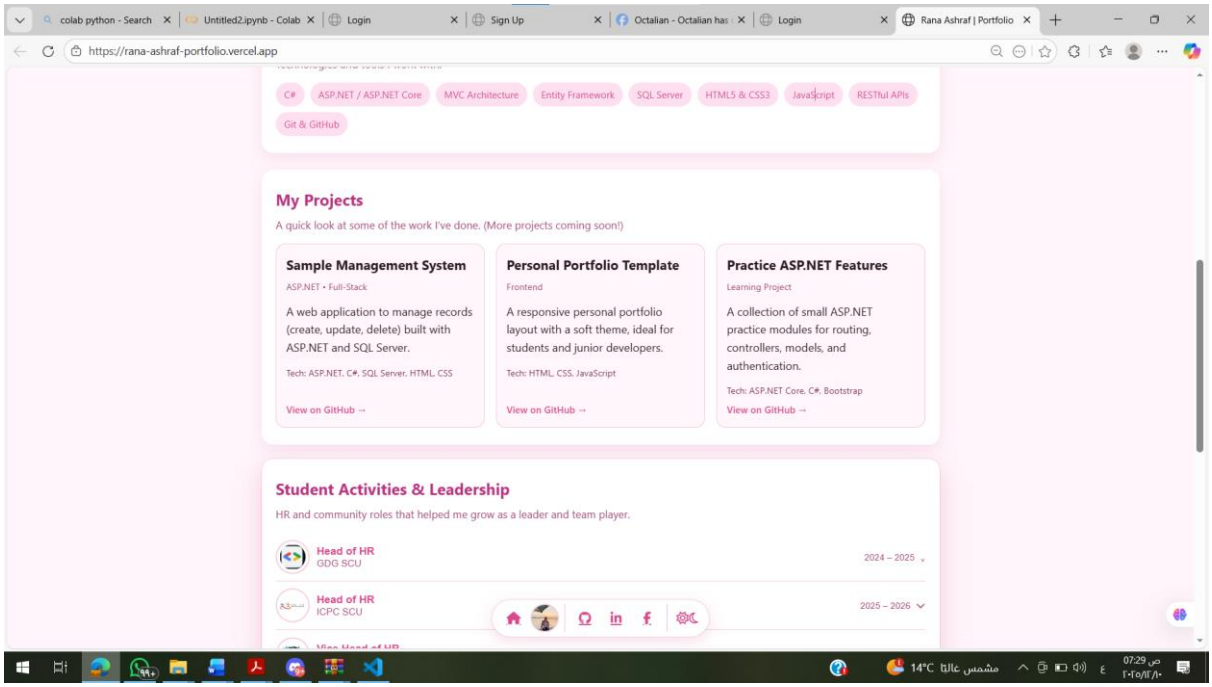
PROBLEMS 2 | OUTPUT | DEBUG CONSOLE | TERMINAL | PORTS

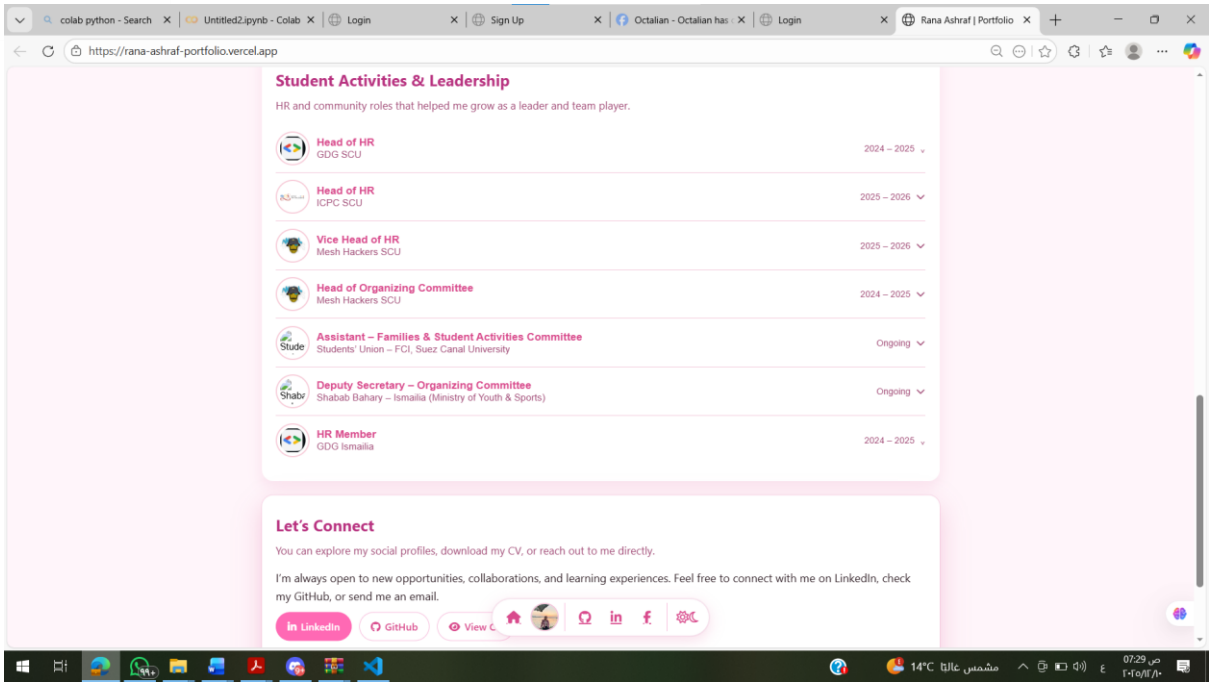
```
PS H:\flask_auth_project> python amazon.py
PS H:\flask_auth_project>
```

portfolio

الاسم:رنا اشرف محمد سعيد(2)






colab python - Search
Untitled2.ipynb - Colab
Login
Sign Up
Octalian - Octalian has
Login
Rana Ashraf | Portfolio
https://rana-ashraf-portfolio.vercel.app
Student Activities & Leadership
HR and community roles that helped me grow as a leader and team player.
Head of HR
GDG SCU
2024 – 2025
Head of HR
ICPC SCU
2025 – 2026
Vice Head of HR
Mesh Hackers SCU
2025 – 2026
Head of Organizing Committee
Mesh Hackers SCU
2024 – 2025
Assistant – Families & Student Activities Committee
Students' Union – FCI, Suez Canal University
Ongoing
Deputy Secretary – Organizing Committee
Shabab Bahary – Ismailia (Ministry of Youth & Sports)
Ongoing
HR Member
GDG Ismailia
2024 – 2025
Let's Connect
You can explore my social profiles, download my CV, or reach out to me directly.
I'm always open to new opportunities, collaborations, and learning experiences. Feel free to connect with me on LinkedIn, check my GitHub, or send me an email.
LinkedIn
GitHub
14°C
07:29 ص