Assignment 1

Jupyter Notebook and Python (45 marks/ 45%)

This is an individual assignment with a total score of 45 marks (45% of overall course assessment.

Complete the following tasks in a Jupyter notebook and submit your work (single **Jupyter notebook**) via the submission point on POLITEMall.

Plagiarism is the taking and using the whole or any part of ideas, words or works of others, including contents generated by AI tools and passing it off as one's own work without acknowledgement of the original source.

Please **submit a Plagiarism Declaration** as a separate file during your submission.

Question 1 (5 marks):

It is important to provide documentation for projects to improve clarity. Using **Markdown**, complete the following:

1. Create a **Header Level 1 title** named **"Project Documentation"**.
   Add an **introductory paragraph** describing the purpose of the project.

2. Insert a **table** with 3 columns: **"Module Name"**, **"Description"**, and **"Version"**.
   Populate it with at least 2 rows of example data.

3. Create a **Header Level 2 title** named **"Features"** and list at least 3 features of the project.

4. Create a **Header Level 3 title** named **"Python Code Example"** and include a Python function that adds two numbers.

5. Create a **Header Level 4 title** named **"Mathematical Formula"**, and display the formula for the area of a circle, $A=\pi r2$.

Here is how it will look.

Project Documentation

This project aims to demonstrate the importance of Markdown for clear and structured documentation. It includes an overview of the project's modules, key features, a Python code example, and a mathematical formula.

Features

- User-friendly interface
- Supports multiple programming languages
- Open-source and customizable

Module Name	Description	Version
Auth System	Handles user authentication	1.0
API Handler	Manages external API requests	2.1

Python Code Example

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

print(add_numbers(3, 5))  # Output: 8
```

Mathematical Formula

$$A = \pi r^2$$

Question 2 : Load containers on the ship(Total 8 marks)

You are given a harbor loading area where containers are stacked in a deck. Each container has a **different size (surface area) and weight**. The deck is represented as a **2D list**, where each cell contains a tuple **(container_size, container_weight)**.

A ship is available to load these containers, but it has two constraints:

1. The **total weight** of all loaded containers must not exceed the ship's **maximum weight capacity (maxWeight)**.

2. The **total surface area** occupied by the containers on the ship must not exceed the ship's **available deck space (maxSurface)**.

Containers **cannot be stacked** on the ship; they must be arranged side by side. Your task is to **determine the maximum number of containers** that can be loaded onto the ship while ensuring both the weight and surface constraints are met. (5 marks)

Requirements:

- Implement a function max_containers(deck: list, maxWeight: int, maxSurface: int) -> int that takes the following inputs:
    - deck: A 2D list where each element is a tuple (container_size, container_weight).
    - maxWeight: The maximum weight capacity of the ship.
    - maxSurface: The available surface area on the ship's deck.
- The function should return the maximum number of containers that can be loaded.

Example Input & Output:

```
deck = [
    [(1, 10), (1, 15), (2, 20)],
    [(1, 5), (2, 25), (1, 10)],
    [(2, 30), (1, 10), (1, 5)]
]
maxWeight = 100  # Ship's weight capacity
maxSurface = 6    # Ship's available deck surface area

print(max_containers(deck, maxWeight, maxSurface))
# Expected Output: The maximum number of containers that can be loaded
```

Test the different scenario:

Test Case 1: Basic Case

Input:

```
deck = [
    [(1, 10), (1, 15), (2, 20)],
    [(1, 5), (2, 25), (1, 10)],
    [(2, 30), (1, 10), (1, 5)]
]
maxWeight = 100
maxSurface = 6
```

Test Case 2: Limited Weight Capacity

Input:

```
deck = [
    [(1, 10), (1, 15), (1, 10)],
    [(2, 30), (2, 25), (1, 10)]
]
maxWeight = 40
maxSurface = 5
```

Test Case 3: Limited Surface Area

Input:

```
deck = [
    [(1, 5), (2, 10), (3, 15)],
    [(1, 5), (2, 10), (3, 15)]
]
maxWeight = 100
maxSurface = 4
```

Test Case 4: No Containers Fit

Input:

```
deck = [
    [(2, 50), (3, 60)],
    [(2, 70), (3, 80)]
]
maxWeight = 40
maxSurface = 3
```

Test Case 5: Large Deck with Optimal Packing

Input:

```
deck = [
    [(1, 5), (1, 5), (1, 5), (1, 5)],
    [(1, 5), (1, 5), (1, 5), (1, 5)],
    [(2, 10), (2, 10), (2, 10), (2, 10)],
    [(3, 15), (3, 15), (3, 15), (3, 15)]
]
maxWeight = 50
maxSurface = 8
```

Question 3: Working with DataFrame (8 marks)

Follow the general steps to obtain the sample output.

fruits_details.xlsx

Step 1: Read in the "fruits_details.xlsx" into a DataFrame.

Step 2: Define the following tables of data as 2 DataFrames without using file read.

February

Quantity Sold	Product	Unit Price
900	Apples	$0.80
300	Bananas	$0.50

March

Quantity Sold	Product	Unit Price
200	Apples	$0.80
100	Bananas	$1.00

Step 3: Concatenate the three DataFrames from Step 1 and Step 2 into a single DataFrame.

Step 4: Add a column to store the unit price with tax. Fruits costing $2 and above are charged with an additional goods tax of 9% while the other fruits will enjoy a lower goods tax rate of 5% instead.

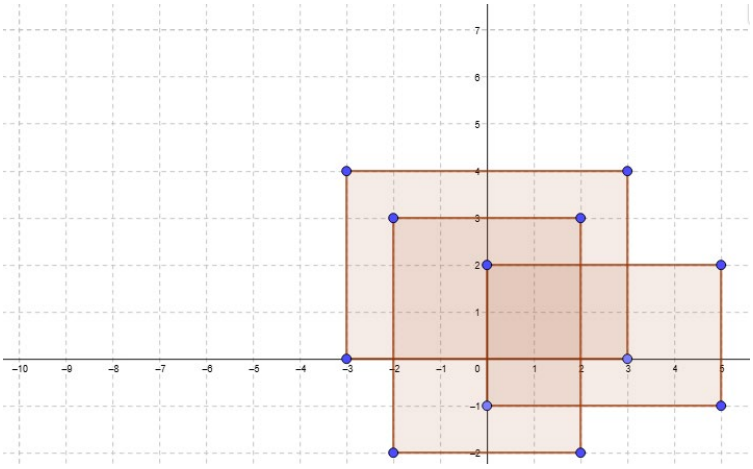
Step 5: Display the final DataFrame and ensure the prices are reflected with a prefix of "S$".

Required Output:

	Quantity Sold	Product	Unit Price	Month	w Tax
0	500	Apples	S$0.80	Jan	S$0.84
1	600	Oranges	S$1.00	Jan	S$1.05
2	500	Durians	S$30.00	Jan	S$32.70
3	400	Blue berries	S$3.00	Jan	S$3.27
4	600	Banans	S$0.50	Jan	S$0.53
5	900	Apples	S$0.80	Feb	S$0.84
6	300	Banana	S$0.50	Feb	S$0.53
7	200	Apples	S$0.80	Mar	S$0.84
8	100	Bananas	S$1.00	Mar	S$1.05

Question 4: Find all overlapped area among the given rectangles(10 marks)

Given the coordinates of N rectilinear rectangles in a 2D plane. Rectangles may overlap, and the overlapping regions should not be counted multiple times. Each pair of rectangle overlap can only count once. Each rectangle is represented by four integers: (xi1, yi1, xi2, yi2), where (xi1, yi1) is the bottom-left corner and (xi2, yi2) is the top-right corner.
Write a python function to return all overlap areas, give an N number of rectangles.



```
# Example test cases
rectangles1 = [
    [-3, 0, 3, 4],
    [0, -1, 9, 2]
]
rectangles2 = [
    [-2, -2, 2, 2],
    [-2, -2, 2, 2],
    [1, 1, 3, 3]
]
rectangles3 = [
    [-3, 0, 3, 4],
    [0, -1, 5, 2],
    [-2, -2, 2, 3]
```

```
]
print(find_overlap_areas(rectangles1))  # Output: Overlap areas for each pair of
rectangles
print(find_overlap_areas(rectangles2))  # Output: Overlap areas for each pair of
rectangles
print(find_overlap_areas(rectangles3))
```

Question 5 (9 marks):

Your task is to design a heater system with a fixed warming radius to ensure all houses receive sufficient heat.

Each house is considered warmed if it falls within the coverage range of at least one heater.

Given the positions of houses and heaters along a one-dimensional line, determine the minimum required radius for the heaters so that every house is covered.

Note: All heaters operate with the same radius, and the goal is to find the smallest possible value that guarantees full coverage.

Example 1:

Input: houses = [1,2,3], heaters = [2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.
Example 2:

Input: houses = [1,2,3,4], heaters = [1,4]
Output: 1
Explanation: The two heaters were placed at positions 1 and 4. We need to use a radius 1 standard, then all the houses can be warmed.
Example 3:

Input: houses = [1,5], heaters = [2]
Output: 3

Question 6 (5 marks):

You need to do a 10 mins presentation of the above Question's code. Describe the key ideas of how your python code is designed to answer the question.

Criteria	Excellent	Good	Satisfactory	Need Improvement
Presentation (5 marks)	**(>4.0 – 5.0)** Presentation ideas are presented clearly; messages are concise, well-articulated, and easy to understand.	**(>3.0 – 4.0)** Presented generally with clear message; most ideas are articulated well with minor confusions.	**(>2.5 – 3.0)** Some presented ideas are unclear; overall message is hard to follow at times.	**(0.0 – 2.5)** Most presentations are confusing or incoherent message; ideas are poorly articulated.