Mark as done

Description Submission view

Available from: Wednesday, 26 November 2025, 9:15 AM
Due date: Wednesday, 26 November 2025, 10:45 AM
Requested files: p1.py, p2.py, p1_input.txt, p2_input.txt, p1_points_1.csv, p1_points_2.csv, p2_items_1.csv, p2_items_2.csv (Download)
Maximum number of files: 9
Type of work: Individual work

Problem 1

Description

Given a finite set of points in a 2D Euclidean plane, the *Closest Pair Problem* asks us to find the two distinct points whose *Euclidean* distance is strictly minimal among all possible pairs in the set.

Your task is to implement a function, `closest_pair` that reads a dataset of 2D Cartesian coordinates from a CSV file and calculates the **minimum euclidean distance** between any two points using a Brute Force approach.

```
def closest_pair(filePath: str) -> float
```

Input Format

- Path of CSV file (`filePath` provided as a string argument).
- Each row in the file represents a point and contains two numerical values(*float/integer*) separated by a comma representing the coordinate: *(x, y)*.

Output Format

Return a single *float* representing the minimum Euclidean distance found.

Examples

Example 1

Input File (*points.csv*):

```
0,0
3,4
10,10
1,0
```

Expected Output:

```
1.0
```

Explanation:

Distance between (0,0) and (3,4) is 5.0.
Distance between (0,0) and (1,0) is 1.0.
Distance between (3,4) and (1,0) is 4.47.
...
(Other pairs are further apart).

Testing Instructions

You can test your code locally using the provided runner.

- The file `p1_input.txt` contains the list of CSV files (e.g., `p1_points_1.csv`, `p1_points_2.csv`) that will be passed as the `filePath` argument.
- You can modify the content of `p1_points_1.csv/p1_points_2.csv` to test on different collection of points.

Restrictions: No *imports* are allowed

?

Problem 2

Description

You are given a budget and a list of items, where each item has a name and a price. Your goal is to select a subset of these items such that the sum of their ***prices* <= *budget***, and the total **price is *maximized***.

You must implement a function, `shop_smart` that reads the items from a CSV file located at `filePath`, and a given `budget`*(float)*, and computes the optimal subset, and writes the names of the chosen items and the final total to an output file named `plan.txt`.

Input Format

- CSV File located at `filePath`:
  - Contains a header row: name,price.
  - Subsequent rows contain the item name (string) and price (float), separated by `','`.
- `budget`: A floating-point number representing the maximum allowable total.

Requirements

1. You should find the subset that maximizes the total spend without exceeding the budget.
2. Ties: If multiple subsets yield the same maximum total, any valid subset is acceptable.

Output File (plan.txt):

1. List the names of the selected items (one per line).
2. The last line must start with `Total:` followed by the numeric sum.

Function Signature

```
def shop_smart(filePath: str, budget: float) -> None:
```

Examples

Example 1

- CSV File (located at `filePath`, e.g., *items.csv*):

```
name,price
Apple,1.0
Banana,2.0
Cherry,3.0
Date,4.0
```

- budget: `6.0`

Expected Output (plan.txt):

Date
Banana
Total: 6.0

Explanation:

- Option A: Date (4.0) + Banana (2.0) = 6.0
- Option B: Apple (1.0) + Banana (2.0) + Cherry (3.0) = 6.0
  Either option is valid as they maximize the utilization to 6.0

Example 2

- CSV File (located at `filePath`, e.g., *items.csv*):

```
name,price
ItemA,4.0
ItemB,3.0
ItemC,3.0
```

- budget: `6.0`

Expected Output (plan.txt):

```
ItemB
ItemC
Total: 6.0
```

Explanation:

- Pick ItemB (3.0) + ItemC (3.0). Total = 6.0.

Testing Instructions

You can test your code locally using the provided runner.

- The file `p2_input.txt` contains the test cases. Each line provides the arguments: a CSV filename (e.g., `p2_items_1.csv`) and a budget (e.g., `60.0`).
- You can change the content of the `p2_items_1.csv/p2_items_2.csv` or the budget value(s) in `p2_input.txt`.
- On running your program, the runner will display the content of the generated `plan.txt` for verification.

Restrictions: No *imports* are allowed

Requested files

p1.py

```
# Follow the problem 1 description strictly to define the closest pair function
```

p2.py

```
# Follow the problem 2 description strictly to define the shop smart function
```

p1_input.txt

```
2
p1_points_1.csv
p2_points_2.csv
```

p2_input.txt

```
2
p2_items_1.csv 60.0
p2_items_2.csv 100.0
```

p1_points_1.csv

```
45.98,23.27
79.37,78.81
89.42,88.92
39.41,-68.85
-95.22,49.83
4.71,52.77
4,5
5,4
```

p1_points_2.csv

```
45.98,23.27
79.37,78.81
89.42,88.92
39.41,68.85
65.6,85.9
86.35,0.41
58.91,33.54
67.9,9.49
95.22,49.83
4.71,52.77
53.71,90.3
83.83,34.77
15.8,68.53
38.59,45.68
71.54,94.38
79.37,78.81
89.42,88.92
39.41,68.85
65.6,-85.9
-86.35,0.41
58.91,33.54
67.9,9.49
95.22,49.83
4.71,52.77
53.71,90.3
83.83,-34.7
0.0,0.0
0.0001,0.00001
```

p2_items_1.csv

```
name,price
Apple,10
Banana,5
Bread,3
Chair,299
Headphones,40
Notebook,10
Advice,0
```

p2_items_2.csv

```
name,price
USB Cable,8
Keyboard,25
Gaming Mouse,40
Desk-Lamp,15
Water Bottle,12

```

VPL