Simulation 11

Programming Techniques · Home Practice

Assessment

Full exam: 6 exercises, 4 + 4 + 2 + 4 + 4 + 2 = 20 points

Partial I: exercises 1·3, 7 + 7 + 6 = 20 points

Partial II: exercises 4·6, 8 + 8 + 4 = 20 points

Instructions

Write independent Python solutions. Use only tools covered in the course. Read the requirements, assumptions and expected results of each exercise before beginning. Input files mentioned by name are included in the accompanying data folder. Each exercise is independent; no solution from another exercise is needed. The reference values illustrate the required behavior, but solutions must work with other valid inputs. No additional validation, formatting or functionality is required beyond what is explicitly requested.

The six exercises follow on the next pages.

Simulation 11

Full exam: 4 + 4 + 2 + 4 + 4 + 2 points. **Partial I:** 7 + 7 + 6. **Partial II:** 8 + 8 + 4.

Exercise 1 – Keyword matches in route descriptions (4 points – full exam)

Partial I value: 7 points

A transit service checks which short words appear in its route descriptions.

For every search keyword, inspect all route descriptions and count the descriptions containing that keyword. Sum these keyword-specific counts for the overall match total. One description may match several keywords, so the overall total is not necessarily the number of descriptions.

Input and assumptions

The route descriptions and words are strings. The words are distinct and nonempty. Matching is case-sensitive and uses substrings.

Required results and operations

a) Count, for each search word, how many route descriptions contain that word
b) Use two nested for loops to compare every word with every description
c) Store the counts in a dictionary and print them sorted by word
d) Print the sum of the individual counts

Tools and restrictions Use standard Python collections, functions when requested, and two genuinely nested for loops. No files or third-party libraries are needed in this exercise.

Reference example

```
routes = ['blue bus to station', 'green bus to garden', 'tram to station']
words = ['bus', 'station', 'garden']

Expected output:
bus: 2
garden: 1
station: 2
Total matches: 5
```

Exercise 2 – Progressive totals (4 points – full exam)

Partial I value: 7 points

A stream of daily expenses must be summarized without losing the order in which they were recorded.

The sequence is processed in order and each reported total must include the current element plus all preceding ones. Preserve the number and order of results; a total for a later position must not be calculated from only that later value.

Input and assumptions

All entries are integers and may be negative; an empty input gives an empty list and count zero. The program must work with other valid data, not only this reference example.

Required results and operations

a) Define `running_totals(values)` returning the sum after each successive entry
b) Define `count_positive(values)` returning the number of strictly positive entries
c) Call the two functions and print their results

Tools and restrictions Use loops, dictionaries, lists and ordinary functions from Lectures 1–10. Do not use file input, NumPy, pandas or extra validation.

Reference example

```
values = [3, 5, -2, 4]

Expected output:
[3, 8, 6, 10]
3
```

Exercise 3 – Find and rename a CSV filename recursively (2 points – full exam)

Partial I value: 6 points

A folder listing is represented by nested lists of filenames. Locate the first CSV filename and create a modified version of its name by replacing hyphens with underscores. Write two small recursive functions for these two tasks.

The first recursive function returns the first filename ending in '.csv' encountered anywhere in the nested lists. Only after finding a name should the second function transform that returned string, replacing each hyphen with an underscore while leaving other characters unchanged. If no filename matches, the search function returns `None`; do not call the transformation function and no `Updated` line is printed.

Required results and operations

a) Define `find_first_csv(data)` to search through nested lists and return the first string ending with `.csv`; return `None` if no matching string exists
b) Define `replace_hyphens(text)` to return a new string in which every `-` character is changed to `_`; implement this transformation recursively, processing the characters of the string
c) Apply the second function only when the first function finds a filename; print both the original match and the modified filename

Reference example

```
data = [
    ['notes.txt', 'sales-2026-final.csv'],
    'archive.csv',
    ['report.csv', ['log.txt']]
]

Expected output:
Found: sales-2026-final.csv
Updated: sales_2026_final.csv
```

Exercise 4 – Event type totals from JSON (4 points – full exam)

Partial II value: 8 points

An event organizer stores registrations in `events.json`. The JSON object contains a list called `events`; each entry has an event `type` and an integer `count`. A type may occur on multiple entries for different sessions.

Build a report with the total count for each distinct event type by combining all entries bearing the same type. Display the results in alphabetical order by type. If the source file cannot be found, print `Input file not found` and do not print the report. The JSON structure is valid whenever the file exists.

Input and assumptions

The supplied events.json contains a JSON object with an events field holding a list. Each entry in that list is an object containing a type string and an integer count. Several entries may share the same type

Combine counts for matching event types and print one line for each distinct type in alphabetical order, formatted as type: total. Use all entries rather than only the first occurrence of a type

Required results and operations

a) Load `events.json` with the standard `json` module and obtain the list stored under `events`
b) Calculate the combined `count` for each distinct event `type`

c) Print each type and its combined count in alphabetical order
d) If the file does not exist, catch `FileNotFoundError` and print `Input file not found`

Tools and errors: Read the file with the standard-library json module; do not use pandas. If the JSON file cannot be opened because it does not exist, display exactly Input file not found without a traceback

Reference example

```
Input file (events.json):
{
  "events": [
    {
      "type": "walk",
      "count": 4
    },
    {
      "type": "talk",
      "count": 4
    },
    {
      "type": "walk",
      "count": 1
    }
  ]
}

Expected result:
talk: 4
walk: 5
```

Exercise 5 – Modular filtering of JSON numbers (4 points – full exam)

Partial II value: 8 points

A sensor saves a list of integer measurements in a JSON file. The report should contain only measurements strictly greater than a prescribed threshold, preserving their original order.

The helper receives numerical values from the loaded JSON list and keeps only those meeting its condition. main.py remains responsible for loading the JSON and displaying or storing the result. Preserve input order and do not require user interaction.

Input and assumptions The supplied input.json contains a dictionary with a numbers list of valid integers. Assume the file is present and correctly formatted.

Select only numbers strictly greater than 5, preserving their input order.

Required results and operations

a) Create helpers.py containing select_above()
b) Load input.json in main.py and call the imported helper
c) Select only numbers strictly greater than 5, preserving their input order.
d) Save the result to selected.json as the JSON field selected and print the result

Tools and restrictions: Use standard json library and a separate helper module. No additional validation is needed.

Reference example

```
input.json:
{"numbers": [5, 9, 6]}

Expected printed result:
[9, 6]
```

Exercise 6 – Recipe ingredient list (2 points – full exam)

Partial II value: 4 points

A recipe maintains an ordered list of ingredients.

One recipe object owns its ingredient list. Adding an ingredient updates that object's stored list, and the reporting method returns or prints the collection as specified. Do not share the same list between unrelated recipe instances.

Input and assumptions

Ingredients are nonempty strings; duplicates may be kept. No exception handling or external data files are required.

Required results and operations

a) Define `Recipe` with name and a new empty ingredient list per instance
b) Implement `.add(ingredient)` and `.size()`
c) Create a recipe, add two ingredients and print the list and its length

Tools and restrictions Use class definitions, instance attributes, simple methods and instance creation from Lecture 18–19.

Reference example

```
Example usage:
recipe_soup = Recipe('Soup')
recipe_soup.add('carrot')
recipe_soup.add('onion')
print(recipe_soup.ingredients)
print(recipe_soup.size())

Expected output:
['carrot', 'onion']
2
```