Simulation 19

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 19

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

Exercise 1 – CSV files in archived groups (4 points – full exam)

Partial I value: 7 points

An archive stores a list of filename groups. Report all CSV filenames regardless of the group in which they appear.

Every group contains a sequence of filenames. Return one flat list with matching '.csv' filenames from all groups, keeping group order and filename order within each group. Keep duplicates if present. The second reported result is the length of that list.

Input and assumptions

Groups are lists of strings, which may be empty. Matching is case-sensitive.

Required results and operations

a) Build a flat list of every filename ending in .csv
b) Use two nested for loops to visit all groups and their filenames
c) Keep the original encounter order and do not remove duplicates
d) Print the list and the total count

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

```
groups = [['budget.csv', 'readme.txt'], ['sales.csv', 'budget.csv'], []]

Expected output:
CSV files: ['budget.csv', 'sales.csv', 'budget.csv']
Count: 3
```

Exercise 2 – Compare measurements in pairs (4 points – full exam)

Partial I value: 7 points

Two sensors measure the same number of observations.

The lists contain readings aligned by index. Report indexes where the first sensor exceeds the second, and separately compute the greatest absolute difference across corresponding readings. Do not compare every value of one sensor against every value of the other.

Input and assumptions

Lists have the same length and contain numeric values; equal values do not count as greater. The program must work with other valid data, not only this reference example.

Required results and operations

a) Define `greater_indices(a, b)` returning indexes where the first list has a strictly larger value
b) Define `largest_gap(a, b)` returning the maximum absolute difference at corresponding positions
c) Print both 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

```
a = [3, 10, 6, 2]
b = [5, 7, 6, 8]

Expected output:
[1]
6
```

Exercise 3 – Find the largest measurement (2 points – full exam)

Partial I value: 6 points

Measurements are stored in nested lists. Return the largest integer, even when some inner lists are empty.

Find the maximum integer among all numeric leaves of the nested lists. Empty sublists must not cause an error or be mistaken for numbers. Return the integer maximum, not its location; follow the stated assumption about whether at least one number exists.

Input and assumptions

The outer object is a list; elements are either integers or other nested lists. Empty lists may occur. Return None if all lists are empty; do not assume the numbers are positive.

Required results and operations

a) Define `nested_max(data)` and make it return the result described above for the entire structure
b) Process nested lists at every depth; empty lists must be handled correctly
c) Call the function(s) on the reference data and print the returned result(s) as shown

Tools and restrictions Use ordinary recursion, conditions, loops when processing nested containers, and simple string/list/dictionary operations taught in Lectures 1–10. Do not use flattening libraries or `isinstance()`; `type(x) == list` or `type(x) == dict` is sufficient.

Reference example

```
data = [-2, [5, [9]], [4]]

print(nested_max(data))

Expected output:
9
```

Exercise 4 – Combine event totals from JSON (4 points – full exam)

Partial II value: 8 points

An event organizer keeps a structured file `events.json` containing a list of records under the key `events`. Every record has an event `type` and its integer participation `count`. Repeated types correspond to separate event sessions.

Prepare a report with the total participation for each type by adding its counts across all sessions. Print the event types alphabetically with their totals. If the input file is missing, handle `FileNotFoundError` and print `Input file not found`; otherwise assume the JSON has the expected structure.

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) Sum the `count` values for all records with the same event `type`
c) Print the combined counts by type in alphabetical order
d) If the file is absent, 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": 3
    },
    {
      "type": "walk",
      "count": 4
    }
  ]
}

Expected result:
talk: 3
walk: 8
```

Exercise 5 – Modular summary of JSON values (4 points – full exam)

Partial II value: 8 points

A JSON file stores integer readings. Use an imported function to return their total and count.

The imported helper returns a two-item summary of the numeric list. Ensure the positions in the list match the specified meaning: first the count, then the sum. main.py reads JSON, prints the list and writes it under the required 'summary' key.

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.

Compute a two-element list containing the number of values and their sum, in that order.

Required results and operations

a) Create helpers.py containing summary()
b) Load input.json in main.py and call the imported helper
c) Compute a two-element list containing the number of values and their sum, in that order.
d) Save the result to summary.json as the JSON field summary 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": [4, 9, 2]}

Expected printed result:
[3, 15]
```

Exercise 6 – Storage box capacity (2 points – full exam)

Partial II value: 4 points

A storage box tracks occupied slots and available capacity.

The box starts with a known capacity and zero used slots. Placing items succeeds only when enough free slots remain, and a rejected request must not change the stored number of used slots. Print both the success flag and the final stored quantity after the user enters a valid integer.

Input and assumptions

The user enters a valid positive integer; no conversion-error handling or retries are needed. No external data files are required.

Required results and operations

a) Define `StorageBox` with label, capacity and used slots initially zero
b) Implement `.place(units)`, returning `False` without changing the state if the capacity would be exceeded, otherwise updating it and returning `True`
c) Create a box with capacity 10, ask the user for the quantity to place using input(), convert it to int and print whether placement succeeded and the slots used

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

Reference example

```
Example usage:
box_A = StorageBox('A', 10)
Input when prompted: 6

Expected output:
True
6
```