Simulation 03

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 03

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

Exercise 1 – CSV files by project (4 points – full exam)

Partial I value: 7 points

Different projects keep lists of filenames. A coordinator wants to know how many CSV files occur in each project.

The input groups filenames by project. Count the names ending exactly in '.csv' for each project, including projects with zero matches. The overall count is the total of all matching filenames across projects, counting repeated filenames separately if they occur in different groups. Display projects in alphabetical order.

Input and assumptions

The input is a dictionary of lists of strings. Matching is case-sensitive, so only the exact lowercase suffix .csv counts. Empty project lists are permitted.

Required results and operations

a) Create a dictionary mapping each project to the number of filenames ending in .csv
b) Use two nested for loops to examine the project groups and their filenames
c) Print the count for each project in alphabetical order and the overall number of CSV files
d) Preserve the original input dictionary

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

```
projects = {'A': ['notes.txt', 'sample.csv', 'log.csv'], 'B': ['image.png', 'table.csv']}

Expected output:
A: 2
B: 1
Total CSV files: 3
```

Exercise 2 – Find and format the first archive filename (4 points – full exam)

Partial I value: 7 points

A research office keeps a list of filenames received from several teams. Some filenames refer to archive reports and others to unrelated documents. An archive report is recognized by having the prefix `ARC-` at the beginning and the extension `.txt` at the end. Both conditions must hold for the same filename.

The office needs to identify the *first* archive report in the given order, rather than collecting or counting every matching file. After finding it, the program should also provide a more readable version of its filename in which every hyphen has been replaced with an underscore. If no filename qualifies, the search result is `None` and there is no filename to reformat.

Input and assumptions

The input is a list of strings, possibly empty. Filenames are case-sensitive and may be repeated. Only the filename strings are processed; the program must not open any files.

Required results and operations

a) Define `first_archive(files)`, returning the first filename that starts with `ARC-` and ends with `.txt`, or `None` if no such filename exists
b) Use a `while` loop to inspect the input filenames in their original order, stopping when the first match has been found
c) For a matching filename, produce a formatted string by replacing every hyphen with an underscore without modifying the original list

d) Print the found filename and its formatted version using the labels in the example; if there is no match, print `First archive: None` and omit the formatted line

Tools and restrictions Use strings, Boolean expressions, a `while` loop and string methods already taught. No filesystem access or external libraries are required.

Reference example

```
files = ['OLD-7.txt', 'ARC-3.tmp', 'ARC-12.txt', 'ARC-99.txt']

Expected output:
First archive: ARC-12.txt
Formatted: ARC_12.txt
```

Exercise 3 – Collect CSV filenames (2 points – full exam)

Partial I value: 6 points

A folder inventory is stored as nested lists of filenames. Return a new flat list containing every filename ending in .csv.

Return the matching filenames themselves in a new list, not a number. Search all nested sublists, regardless of depth, and preserve the left-to-right order in which names appear. Keep duplicate filenames if they appear more than once. A structure without '.csv' names returns an empty list.

Input and assumptions

The outer object is a list; elements are either strings or other nested lists. Empty lists may occur. Preserve the left-to-right order, include repeated filenames and return an empty list if none match.

Required results and operations

a) Define `collect_csv(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 = ['notes.txt', ['sales.csv', 'report.pdf'], ['prices.csv', ['summary.csv']]]

print(collect_csv(data))

Expected output:
['sales.csv', 'prices.csv', 'summary.csv']
```

Exercise 4 – Filtering events from JSON (4 points – full exam)

Partial II value: 8 points

An event log uses a JSON array of records, each with a category and a participation count. Report the categories whose total count reaches a threshold.

The JSON file contains several event records; a repeated event type contributes additional participants to that type. Decide which types reach the threshold only after combining all their counts. The final report lists qualifying types alphabetically. Only a missing input file is handled as an exception.

Input and assumptions

The JSON document is a valid array; counts are nonnegative integers. A missing category in the result is not printed. The file may contain no events.

Required results and operations

a) Read events.json using the standard json module, where each object has type and count
b) Add counts for repeated types and retain only types with a combined count of at least 5
c) Print each selected type and total in alphabetical order
d) Handle FileNotFoundError if events.json does not exist by printing Input file not found

Tools and restrictions Use json.load(), dictionaries and a single FileNotFoundError handler. No pandas.

Reference example

```
events.json:
[{"type": "talk", "count": 3}, {"type": "tour", "count": 2}, {"type": "talk", "count": 4}]

Expected output:
talk 7
```

Exercise 5 – Transform a JSON list with a helper module (4 points – full exam)

Partial II value: 8 points

A small monitoring program stores numerical measurements in `input.json`, a JSON object containing a list under the key `numbers`. The program must multiply each number in that list by a fixed factor of 4. For example, an input value of 6 becomes 24; the position of each number in the list must not change.

Separate the numerical operation from file handling. The function `multiply(values, factor)` belongs in `helpers.py` and returns a new list of transformed numbers. The main program imports the function, loads the original list from JSON, prints the transformed list, and writes it to `output.json` under the key `result`. Do not modify the input file.

Input and assumptions The supplied input.json contains a JSON object with a numbers field holding a list of numeric values. The multiplication factor is the fixed value specified in the requirements; assume the file exists and is valid JSON

Provide two scripts: helpers.py defines multiply(values, factor), and main.py loads the data and imports that function. Write output.json as an object containing the key result and the transformed list. Print the same list to the terminal, keeping the original element order

Required results and operations

a) Create helpers.py containing multiply(values, factor), returning a new list with each value multiplied by factor
b) Create main.py importing this function
c) Load the numbers list from input.json
d) Write the transformed list to output.json under the key result
e) Use a factor of 4 and print the resulting list

Tools and restrictions: Use the standard-library json module and modular Python code. The helper should return a new list rather than modify the input. Do not use pandas or NumPy, and no additional exception handling is required

Reference example

```
Input file:
{
  "numbers": [
    6,
    9,
    4
  ]
}

Expected printed result:
[24, 36, 16]
```

Exercise 6 – Temperature log (2 points – full exam)

Partial II value: 4 points

A small weather device stores the last recorded temperature.

The object stores one latest temperature. Setting a new temperature replaces the previous value rather than adding to it. The Boolean method reports whether the current stored temperature is at most zero. Print its state after updating the same instance.

Input and assumptions

All readings are numbers and need no input validation. No exception handling or external data files are required.

Required results and operations

a) Define `TemperatureLog` with location and temperature attributes
b) Implement `.update(reading)` to replace the stored temperature and `.is_freezing()` to test whether it is at most zero
c) Update a test object and print its temperature and Boolean status

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

Reference example

```
Example usage:
sensor_port = TemperatureLog('Port', 4)
sensor_port.update(-2)
print(sensor_port.temperature)
print(sensor_port.is_freezing())

Expected output:
-2
True
```