Simulation 13

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 13

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

Exercise 1 – Select text files from folder lists (4 points – full exam)

Partial I value: 7 points

A research assistant has a list of folders; each folder is represented by a list of filenames.

Each folder is represented by a list of filename strings. The result contains only names ending in '.txt', in folder order and then filename order. Identical names in different folders remain separate entries. Print both the selected list and its length; do not open the files themselves.

Input and assumptions

The input is a list of lists of strings. Empty folders are allowed. The match is case-sensitive, so .TXT does not count.

Required results and operations

a) Create a new list of all names ending exactly with .txt
b) Use two nested for loops to inspect every folder and its filenames
c) Keep matching names in encounter order, including duplicates
d) Print the list and the number of selected files

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

```
folders = [['notes.txt', 'image.png'], [], ['data.csv', 'readme.txt', 'notes.txt']]

Expected output:
Text files: ['notes.txt', 'readme.txt', 'notes.txt']
Count: 3
```

Exercise 2 – Stop when a budget is exceeded (4 points – full exam)

Partial I value: 7 points

A club records consecutive purchases. Determine how many purchases can be included without exceeding a fixed budget.

Only an initial sequence of purchases can be accepted. As soon as the next purchase would exceed the budget, stop rather than skipping it and considering later purchases. Return how many purchases were included and their combined cost. The budget may be exactly exhausted.

Input and assumptions

Prices and budget are nonnegative numbers. The list may be empty.

Required results and operations

a) Implement affordable_prefix(prices, budget), returning (count, total) for the longest initial sequence of purchases whose total does not exceed the budget
b) Stop before the first purchase that would exceed the budget
c) Use a while loop; do not skip over a purchase to include later items

Tools and restrictions Use basic Python lists, functions, conditions and the specified loops. No external libraries or file operations are required.

Reference example

```
prices = [4, 6, 9, 2]
budget = 12

Expected output:
Affordable: (2, 10)
```

Exercise 3 – Sum squared deviations (2 points – full exam)

Partial I value: 6 points

A list of groups contains integer deviations, possibly nested several levels deep. Compute the sum of their squares.

The input is a recursively nested collection of integer deviations. For each integer, contribute its square to the final sum, including negative deviations (whose squares are positive). Empty inner groups contribute zero. Return one total number for the entire structure.

Input and assumptions

The outer object is a list; elements are either integers or other nested lists. Empty lists may occur. Negative numbers also contribute positive squares; an empty list contributes zero.

Required results and operations

a) Define `sum_squares(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, [3, [1]], []]

print(sum_squares(data))

Expected output:
14
```

Exercise 4 – Stock items above a threshold (4 points – full exam)

Partial II value: 8 points

A supply store records item quantities in `notes.txt`. Each nonempty line has one item name and one integer quantity; an item may occur several times because its deliveries are recorded separately. The store needs a report of the items whose total available stock is *at least four units*.

A negative quantity is considered an invalid record and must be ignored, while processing continues with the other lines. Define a simple custom exception `NegativeQuantityError` derived from `ValueError`. The function `checked_quantity(text)` converts one quantity and raises this exception when it is negative; the calling program catches it, prints `Skipped: name` for that row and continues. After all valid records have been combined by item, print qualifying names alphabetically with their totals. If the file is missing, print `Input file not found` instead of any report.

Input and assumptions Each nonempty line contains one item name without spaces and one valid integer. A negative quantity represents an invalid record, which must be ignored after printing Skipped: followed by its name. Valid items may repeat. Assume at least one valid record.

Catch NegativeQuantityError for a negative record, print Skipped: name, and continue processing. Print items with valid combined quantity at least 4, alphabetically. If the file is absent, catch FileNotFoundError and print Input file not found instead of a report.

Required results and operations

a) Read notes.txt, combining nonnegative quantities for each item name
b) Produce the requested final selection or report using the combined totals
c) Define class NegativeQuantityError(ValueError) using pass, and a function `checked_quantity(text)` that raises it if the converted integer is negative

Tools and restrictions: Use ordinary text-file reading, a dictionary, raise and try-except. NegativeQuantityError is a simple custom exception subclass; no custom initializer is required. Do not use pandas.

Reference example

```
notes.txt:
paper 2
pens 4
folders 3
paper 1
markers -2

Expected printed result:
Skipped: markers
pens: 4
```

Exercise 5 – Convert workshop durations to minutes (4 points – full exam)

Partial II value: 8 points

A makerspace records the length of workshops in whole hours. Prepare a schedule containing each workshop duration in minutes. The imported function processes all records and returns the transformed results. The main script handles CSV input and output, and displays the results.

Input and assumptions All input quantities are valid nonnegative integers. Repeated identifiers are allowed and must be processed separately. Preserve original row order.

Required results and operations

a) Create `time_tools.py` containing `convert_durations(records)`, which iterates over all input records and returns a list of transformed pairs
b) In `main.py`, read `sessions.csv` with the standard `csv` module, converting the numeric fields to integers, and call the imported function once
c) For each record, multiply the duration in hours by 60 to obtain its equivalent in minutes
d) Write `minutes.csv` with header `session,minutes` and print the returned values in the reference format

Tools and restrictions: Use ordinary functions, loops, modules, CSV reading and writing. Do not use pandas or external libraries.

Reference example

```
sessions.csv:
session,hours
Intro,2
Design,1
Practice,3

minutes.csv:
session,minutes
Intro,120
Design,60
Practice,180

Expected printed result:
Intro: 120
Design: 60
Practice: 180
```

Exercise 6 – Scoreboard update (2 points – full exam)

Partial II value: 4 points

A tournament scoreboard tracks the points for one team.

The scoreboard owns its current score as instance state. After constructing the object, apply the specified change through a method, then display the updated score. Do not substitute an external variable for the instance attribute.

Input and assumptions

No tie-breaking or input validation is needed. No exception handling or external data files are required.

Required results and operations

a) Create `Scoreboard` storing team and current points
b) Define `.add(amount)` and `.ahead_of(other)` comparing the scores of two objects
c) Create two scoreboards, add points to the first and print the resulting score and comparison

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

Reference example

```
Example usage:
score_red = Scoreboard('Red', 6)
score_blue = Scoreboard('Blue', 8)
score_red.add(4)
print(score_red.points)
print(score_red.ahead_of(score_blue))

Expected output:
10
True
```