Simulation 17

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 17

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

Exercise 1 – Find repeated product codes (4 points – full exam)

Partial I value: 7 points

A list contains product codes, some of which occur more than once.

A repeated product code must be returned only once even if it occurs three or more times. Output codes in alphabetical order. Codes appearing once are excluded; comparing equal code strings is case-sensitive.

Input and assumptions

Codes are nonempty case-sensitive strings; no file input is needed. The program must work with other valid data, not only this reference example.

Required results and operations

a) Define `repeated_codes(codes)` returning codes occurring at least twice
b) Include each repeated code once and present codes in alphabetical order
c) Print the returned list

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

```
codes = ['A1', 'B2', 'A1', 'C3', 'B2', 'A1']

Expected output:
['A1', 'B2']
```

Exercise 2 – Row totals and row filtering (4 points – full exam)

Partial I value: 7 points

A small table is represented as a list of rows containing numeric measurements. For each row, calculate its total; then build a second list containing the original rows whose totals are at least the specified minimum.

Compute one total per input row, preserving row order. The selected rows are the original row lists whose totals meet the threshold, not lists containing only their totals. Show the totals list and the selection as distinct outputs.

Required results and operations

a) Define `row_totals(table)` returning a list containing the sum of the values in each row, preserving row order
b) Define `rows_above(table, minimum)` returning the original rows whose sum is at least `minimum`
c) Print both resulting lists for the reference input

Reference example

```
table = [[2, 4], [7, 5], [1, 2]]
minimum = 8

Expected output:
[6, 12, 3]
[[7, 5]]
```

Exercise 3 – Count and find an odd value (2 points – full exam)

Partial I value: 6 points

Given a flat list of integers, write two short recursive functions: count odd integers and return the first odd integer.

One recursive function counts all odd values; the other returns the first odd value in the input order, or None if none exists. A list can contain duplicates and zero; the empty list is a valid input. Both functions must be recursive, not a single shared iterative computation.

Input and assumptions

The argument is a flat list of integers, not a nested list. Empty lists may occur. The list is not nested; return None if no odd value occurs.

Required results and operations

a) Define the two separate recursive functions `count_odd(values)` and `first_odd(values)`, each returning its own result
b) Use recursion on the list rather than solving the task with a single loop; handle an empty list
c) Call the function(s) on the reference data and print the returned result(s) as shown

Tools and restrictions Use recursion with list indexing and slicing, not a loop that performs the entire computation. No additional libraries are required.

Reference example

```
values = [4, 8, 3, 7, 2]

print('Odd count:', count_odd(values))
print('First odd:', first_odd(values))

Expected output:
Odd count: 2
First odd: 3
```

Exercise 4 – A text-based category report (4 points – full exam)

Partial II value: 8 points

A repair desk stores category and duration pairs in a text file. Report the total duration per category, keeping only categories above a limit.

Each text-file record contains a category and a duration. Accumulate durations for repeated categories, then print only categories with totals strictly greater than the given limit. Display categories alphabetically, and print the specified message if the file is absent.

Input and assumptions

Categories are single words, durations are valid nonnegative integers and each existing line has exactly two fields. An empty input produces no totals.

Required results and operations

a) Read activities.txt, one category and one integer duration per nonempty line, separated by whitespace
b) Aggregate durations belonging to the same category
c) Print totals strictly above 10, in alphabetical order
d) If the file does not exist, handle FileNotFoundError and print Input file not found

Tools and restrictions Use built-in file operations, a dictionary and one FileNotFoundError handler.

Reference example

```
activities.txt:
check 7
clean 3
check 6
repair 11

Expected output:
check 13
repair 11
```

Exercise 5 – Packaging weight after removing material (4 points – full exam)

Partial II value: 8 points

A shipping laboratory records the gross weight of parcels and the weight of protective packaging. Each row corresponds to one parcel, and the report needs the net weight after packaging is removed. 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 Each row has a parcel identifier and two valid nonnegative integer quantities, with packaging no greater than gross weight. Preserve original row order.

Required results and operations

a) Create `shipping.py` containing `net_weights(records)`, which iterates over all input records and returns a list of transformed pairs
b) In `main.py`, read `parcels.csv` with the standard `csv` module, converting the numeric fields to integers, and call the imported function once
c) For each record, subtract packaging from gross weight to obtain the net weight of the parcel
d) Write `net_weights.csv` with header `parcel,net` 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

```
parcels.csv:
parcel,gross,packaging
P1,9,2
P2,12,4
P3,6,1

net_weights.csv:
parcel,net
P1,7
P2,8
P3,5

Expected printed result:
P1: 7
P2: 8
P3: 5
```

Exercise 6 – Book feedback (2 points – full exam)

Partial II value: 4 points

A book review collects simple integer ratings.

A book review object stores its ratings as its own list. Adding a rating updates that list, and the mean uses the ratings currently stored. An object with no ratings reports the explicitly specified default rather than causing division by zero.

Input and assumptions

Ratings are integers 1–5 and need not be validated. No exception handling or external data files are required.

Required results and operations

a) Define `BookFeedback` with title and an independent empty list of ratings
b) Implement `.add_rating(rating)` and `.average()`, returning 0 for no ratings
c) Create the object, store the two ratings and print the resulting mean

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

Reference example

```
Example usage:
feedback_atlas = BookFeedback('Atlas')
feedback_atlas.add_rating(4)
feedback_atlas.add_rating(5)
print(feedback_atlas.average())

Expected output:
4.5
```