Simulation 10

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 10

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

Exercise 1 – Pairwise comparison of measurements (4 points – full exam)

Partial I value: 7 points

A laboratory compares numerical measurements recorded in two separate experiments.

Compare each element of the first list with each element of the second list, rather than only comparing elements with the same index. The output must distinguish individual qualifying pairs from the total number of such pairs. Preserve encounter order for pair listings and count repeats as separate occurrences.

Input and assumptions Both lists are numeric and have the same length. Empty lists are allowed.

Only corresponding positions are compared, never every possible pair.

Required results and operations

a) Compare values at matching positions in two lists
b) Count the positions where the first reading is greater
c) Count the positions where both readings are equal
d) Print the two counts

Tools and restrictions: Use only Python structures and operations taught in Lectures 1–10; no files, external libraries or additional input validation.

Reference example

```
left = [3,8,5,9]; right = [4,6,5,7]

Expected output:
Left larger: 2
Equal: 1
```

Exercise 2 – Summarize a rectangular table (4 points – full exam)

Partial I value: 7 points

A laboratory stores a numeric table as a list of rows. Compute the sum of each row using indices rather than direct iteration.

Each inner list is a row of the table. The summary must cover every row, including rows that may have no values, while the final aggregate covers the whole table. Do not treat columns as independent groups unless the task specifically requests that.

Input and assumptions

All entries are numeric. Rows may have different lengths.

Required results and operations

a) Implement row_sums(table), returning a list of the row totals in their original order
b) Use an outer while loop for rows and an inner while loop for the elements of each row
c) Return an empty list for an empty table and zero for an empty row

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

Reference example

```
table = [[2, 4, 1], [], [5, 3]]

Expected output:
Row sums: [7, 0, 8]
```

Exercise 3 – Find the first negative adjustment (2 points – full exam)

Partial I value: 6 points

A financial office stores account adjustments in a nested dictionary. Some values are integers, while others are dictionaries containing further adjustment records. The office wants to find the first negative adjustment without knowing how deeply it is nested.

Implement a recursive function that examines the numeric *values* of the dictionaries in their normal insertion order, descending into a nested dictionary when encountered. Return the first negative integer found, not its key, its position or the smallest number. If no negative value exists, return `None`. An empty nested dictionary contains no adjustments.

Input and assumptions

The outer object is a dictionary. Each value is either a number or another nested dictionary. Keys are strings; empty dictionaries may occur. Use dictionary insertion order, descending into each sub-dictionary before continuing; return None when absent.

Required results and operations

a) Define `first_negative(data)` and make it return the result described above for the entire structure
b) Process dictionary values at every nesting level; empty dictionaries 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 = {'north': {'a': 5, 'b': -2}, 'south': {'c': -7}}

print(first_negative(data))

Expected output:
-2
```

Exercise 4 – Sum participant quantities from CSV (4 points – full exam)

Partial II value: 8 points

A registration office stores transaction quantities in `records.csv`. Each row identifies a participant and one integer quantity; the same participant may be listed on several rows. The required report contains one quantity per participant, obtained by adding all that participant's rows.

Read the CSV file with the standard `csv` module, combine quantities by name and display one `name: total` line for each distinct participant, in alphabetical order. Repeated names must not overwrite earlier quantities. Assume the file has the correct header and numeric values.

Input and assumptions Use the supplied records.csv file with a header row name,quantity. Each subsequent row contains a name and a quantity written as an integer. Names may be repeated, and there may be any number of data rows

Compute the sum of quantities associated with each name and print one result per distinct name in alphabetical order. The header is metadata, not a record; retain the exact name spelling present in the file

Required results and operations

a) Read `records.csv` using the standard `csv` module, with columns `name` and `quantity`
b) Sum the quantities across all rows for each distinct name
c) Print `name: total` for each participant in alphabetical order

Tools and restrictions: Use the Python standard-library csv module to read the file and built-in Python data structures for the aggregation. Do not use pandas or NumPy. Assume a readable file with valid columns and numeric quantities unless the task explicitly states otherwise

Reference example

```
Input file (records.csv):
name,quantity
Alice,3
Bob,7
Alice,4
Charlie,2

Expected result:
Alice: 7
Bob: 7
Charlie: 2
```

Exercise 5 – Command-line sign classification (4 points – full exam)

Partial II value: 8 points

A command-line utility receives integer values and uses an imported function to label each one as positive, negative or zero.

The integers are supplied on the command line, not through repeated calls to input(). Each argument must be converted and classified separately, preserving input order. The conversion error behavior is exactly that specified in the task.

Input and assumptions

Assume there is at least one command-line argument. Values may be negative, zero or positive. No special formatting is needed.

Required results and operations

a) Create helpers.py defining sign_label(value), returning the strings positive, negative or zero
b) In main.py, use sys.argv to read the input values and print one label per argument in the same order
c) Catch ValueError for any invalid integer and print Invalid number, without a traceback

Tools and restrictions Use sys.argv, separate Python modules and int() with one try-except ValueError.

Reference example

```
Command: python main.py -2 0 6

Expected output:
negative
zero
positive
```

Exercise 6 – Seat reservation (2 points – full exam)

Partial II value: 4 points

A small venue keeps the number of available seats.

Each instance represents one seating section and stores its number of currently available seats. After creating a section, apply the reservation method and report whether reservation was accepted along with the stored final state. The method, not the caller, must implement the specified rule.

Input and assumptions

Reservation requests are positive integers. No exception handling or external data files are required.

Required results and operations

a) Create `SeatGroup` with section and number of available seats

b) Implement `.reserve(number)` returning success and updating available seats only when sufficient
c) Create one object, make a reservation and print success and seats left

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

Reference example

```
Example usage:
seats_front = SeatGroup('Front', 12)
print(seats_front.reserve(4))
print(seats_front.seats)

Expected output:
True
8
```