Simulation 02

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 02

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

Exercise 1 – Grouping records by owner (4 points – full exam)

Partial I value: 7 points

The museum reception maintains pairs (person, item). Create a dictionary grouping the items recorded for each person.

Treat every (person, item) record as one entry. A person can occur more than once and must have one combined list in the result, while repeated items remain repeated. Print the people alphabetically; the order of items for each person must follow their original appearance in the records.

Input and assumptions Every record is a two-element tuple containing a person and an item, both strings. A person can appear several times.

Each person is printed once with all associated items in encounter order.

Required results and operations

a) Create a dictionary in which each person maps to a list of their items
b) Keep the items in their original record order, including repetitions
c) Print people in alphabetical order, showing their lists
d) Do not modify the input list

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

Reference example

```
records = [('Alice','pen'), ('Bob','book'), ('Alice','paper'), ('Bob','pen')]

Expected output:
Alice: ['pen', 'paper']
Bob: ['book', 'pen']
```

Exercise 2 – First threshold crossing in each group (4 points – full exam)

Partial I value: 7 points

A workshop records daily production counts for several independent production lines. Determine when the running total of each line first reaches its production target.

Treat the groups as independent production lines. For each line, consider the accumulated production up to successive readings and report the first zero-based position at which that line reaches the shared target. If a line never reaches the target, place None in the corresponding output position. Preserve the order of groups.

Input and assumptions

All readings are nonnegative integers and the target is positive. Groups may contain empty lists; do not combine totals from different groups.

Required results and operations

a) Implement group_results(groups, target), returning a list of zero-based indices, one for each group
b) For each group, return the index of the first reading that brings its running total to at least target, or None if it never happens
c) Use a for loop to visit groups and a nested while loop to inspect successive readings within the current group

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

Reference example

```
groups = [[3, 4, 2, 5], [2, 2], [], [10]]
target = 10

Expected output:
First reached: [3, None, None, 0]
```

Exercise 3 – Count small readings in nested records (2 points – full exam)

Partial I value: 6 points

An instrument stores readings in dictionaries grouped by department. Some departments contain further dictionaries. Count readings below a given limit.

Count the numeric leaf readings below the limit, wherever they occur in the nested dictionaries. Dictionary keys identify departments or measurements but must not be counted. Dictionaries can be empty and may contain other dictionaries rather than numbers. The result is one integer for the whole input structure.

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. Only dictionary values representing readings count; dictionary keys are labels, not readings.

Required results and operations

a) Define `count_below(data, limit)` 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 = {'east': {'a': 3, 'b': 11}, 'west': {'unit': {'c': 2}, 'd': 8}}
limit = 5

print(count_below(data, limit))

Expected output:
2
```

Exercise 4 – Totals per participant from a CSV (4 points – full exam)

Partial II value: 8 points

A registration office receives a CSV table in which each row records a name and the quantity associated with one transaction. A person may appear several times because separate transactions are recorded independently. The office needs one combined quantity for each person, rather than one printed result per transaction.

Read the `records.csv` file using the standard `csv` module. Add the quantity values belonging to the same name and display the final totals in alphabetical order. For example, two rows with the same name and quantities 3 and 5 contribute 8 to that person's report. Do not confuse the number of rows with the sum of their numeric quantities.

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) Add the numeric quantities for each name, including repeated names
c) Print each name and its combined quantity 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,4
Alice,5
Charlie,2

Expected result:
Alice: 8
Bob: 4
Charlie: 2
```

Exercise 5 – Command-line rectangle report (4 points – full exam)

Partial II value: 8 points

A program receives the width and height of a rectangle from the command line and delegates the calculation to a function in a separate module.

The arguments are provided when running the script, rather than typed after it starts. The helper returns the rectangle's area and perimeter from the two dimensions; the main script handles argument acquisition and displays both labeled values. The specified error concerns conversion of the provided dimensions, not the mathematical formulas.

Input and assumptions

Assume that exactly two arguments are supplied; positive widths and heights are guaranteed when numeric. No retry loop or range validation is requested.

Required results and operations

a) Create geometry.py defining rectangle(width, height), returning a tuple (area, perimeter)
b) In main.py, use argparse.ArgumentParser to read two positional arguments (width and height), convert them to floats, import rectangle and print the returned results with the labels Area and Perimeter
c) Catch ValueError for a nonnumeric argument and print Invalid number without performing the calculations; keep the execution code in a main() function called from the `__name__ == "__main__"` guard

Tools and restrictions Use the standard-library argparse module, a separate module and try-except ValueError. No third-party packages are required.

Reference example

```
Command: python main.py 4 3

Expected output:
Area: 12.0
Perimeter: 14.0
```

Exercise 6 – Parking ticket duration (2 points – full exam)

Partial II value: 4 points

A parking ticket keeps track of how many paid minutes remain.

The ticket retains its own number of paid minutes. Using some time reduces that stored value, and the validity check refers to the updated minutes remaining. The reference request is guaranteed not to exceed the remaining time; no additional case handling is required.

Input and assumptions

The requested time never exceeds the time remaining. No exception handling or external data files are required.

Required results and operations

a) Define `ParkingTicket` with `code` and `minutes`
b) Implement `.use_time(amount)` and `.is_valid()` returning whether positive time remains
c) Create a ticket, use 15 minutes and print the remaining time and validity

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

Reference example

```
Example usage:
ticket_P42 = ParkingTicket('P42', 40)
ticket_P42.use_time(15)
print(ticket_P42.minutes)
print(ticket_P42.is_valid())

Expected output:
25
True
```