Simulation 18

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 18

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

Exercise 1 – Categorize inspection results (4 points – full exam)

Partial I value: 7 points

A list stores pairs (site, status). Organize site names into groups by status.

The input records are pairs of site and status. Collect the sites under their corresponding status labels; a repeated site may remain repeated if it appears in several records. The result is a mapping from status to lists of site names, in input order within each list.

Input and assumptions

Each record contains exactly a nonempty site name and a status string; duplicates may remain. The program must work with other valid data, not only this reference example.

Required results and operations

a) Define `by_status(records)` returning a dictionary from each status to a list of site names
b) Keep site names in input order within each status group
c) Print the returned dictionary

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

```
records = [('A', 'ok'), ('B', 'review'), ('C', 'ok')]

Expected output:
{'ok': ['A', 'C'], 'review': ['B']}
```

Exercise 2 – Count qualifying cells in a table (4 points – full exam)

Partial I value: 7 points

A small timetable is represented by a list of rows of integers. Count how many cells contain values above a chosen minimum.

A table can have rows of different lengths, including empty rows. Count all individual cells whose numerical value is strictly greater than the minimum, returning one final count for the entire table. The requested two while loops traverse rows and their cells, not recursion.

Input and assumptions

The table is a list of numeric lists. The number of rows and the number of columns may vary.

Required results and operations

a) Implement count_cells(table, minimum), returning the number of entries strictly greater than minimum
b) Use two nested while loops, one for rows and one for values within each row
c) Include all rows, including empty ones

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, 8, 5], [9], [], [1, 7]]
minimum = 5

Expected output:
Qualifying cells: 3
```

Exercise 3 – Select named fields recursively (2 points – full exam)

Partial I value: 6 points

A nested configuration dictionary contains keys whose names may begin with req_. Return a list of all matching key names.

Collect key names starting with 'req_' wherever they appear in a nested dictionary. A matching key contributes its own name even when its associated value is another dictionary. Preserve the order specified in the problem and return one list of matches.

Input and assumptions

The outer object is a dictionary. Each value is either a value or another nested dictionary. Keys are strings; empty dictionaries may occur. Match dictionary keys, not their values; recurse only into values that are dictionaries.

Required results and operations

a) Define `required_keys(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 = {'req_width': 8, 'details': {'req_height': 5, 'color': 'blue'}, 'active': 1}

print(required_keys(data))

Expected output:
['req_width', 'req_height']
```

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

Partial II value: 8 points

A registration desk records quantities from several participants in `records.csv`. Each row contains a participant's name and the quantity registered on one occasion. A participant can have several rows that must be combined for the final report.

Read the file using the standard `csv` module and add together all quantities for the same name. Print one total per distinct participant in alphabetical order, using the form `name: total`. For example, separate rows for Alice with quantities 3 and 6 give a combined result of 9. No conversion-error handling is needed because the CSV is valid.

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) Combine the quantities of rows belonging to the same name
c) Print each name followed by 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,5
Alice,6
Charlie,2

Expected result:
Alice: 9
Bob: 5
Charlie: 2
```

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

Partial II value: 8 points

A command-line utility classifies each supplied integer as even or odd using a reusable function.

Process exactly the supplied command-line integers and print the parity label for each in original order. The helper classifies one converted integer; main.py coordinates the three values. If a conversion fails, display the prescribed error instead of partial classifications.

Input and assumptions Three values are supplied after the Python script name. Every value is intended to be an integer; non-integer text must be handled as specified.

Print even or odd for each integer, preserving argument order.

Required results and operations

a) Implement parity() in helpers.py and import it into main.py
b) Read exactly three command-line arguments as integers using sys.argv
c) Print even or odd for each integer, preserving argument order.
d) If any argument is not a valid integer, catch ValueError, print Invalid integer and do not print partial results

Tools and restrictions: Use Python imports, sys.argv and try-except ValueError; no interactive input() or external packages.

Reference example

```
Example command: python main.py 3 10 15

Expected output:
odd
even
odd
```

Exercise 6 – Shopping basket total (2 points – full exam)

Partial II value: 4 points

A shopping basket stores prices of selected products.

The shopping basket maintains a list of prices associated with that one instance. Adding an item changes the stored list; the total method returns the sum of all prices currently stored, not only the most recently added item.

Input and assumptions

Prices are nonnegative integers; no tax calculation is needed. No exception handling or external data files are required.

Required results and operations

a) Define `Basket` with a label and a fresh empty list of prices
b) Implement `.add_price(price)` and `.total()`

c) Add two prices to a basket and print the total

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

Reference example

```
Example usage:
basket_weekend = Basket('Weekend')
basket_weekend.add_price(7)
basket_weekend.add_price(12)
print(basket_weekend.total())

Expected output:
19
```