Simulation 14

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 14

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

Exercise 1 – Team point totals (4 points – full exam)

Partial I value: 7 points

A competition supplies team-score pairs; the same team may appear more than once.

Each individual team-score record contributes to its team's total. A team appearing several times must have its points accumulated. The output dictionary has one key per distinct team; print the totals alphabetically regardless of the order of the records.

Input and assumptions

Each record is a tuple of a nonempty team string and a nonnegative integer score. The program must work with other valid data, not only this reference example.

Required results and operations

a) Define `team_totals(results)` that sums points for each team in a dictionary
b) Print each team and its total in alphabetical order
c) Do not overwrite an earlier score when the team repeats

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

```
results = [('Red', 4), ('Blue', 3), ('Red', 7), ('Green', 2)]

Expected output:
Blue 3
Green 2
Red 11
```

Exercise 2 – Find a target in each code (4 points – full exam)

Partial I value: 7 points

An archive holds several strings representing item codes. For each code, report the position of its first dash (-).

For every code, return the zero-based position of its first dash, or -1 when the code contains no dash. Do not return all dash positions and do not compare one code with another. The output list contains exactly one position per input code.

Input and assumptions

Codes are strings and may be empty. Character indexing begins at zero.

Required results and operations

a) Implement first_dash_positions(codes), returning one index per code in a new list
b) Use -1 when a code contains no dash
c) Use a for loop to visit the codes and an inner while loop to inspect characters, stopping at the first dash in each code

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

Reference example

```
codes = ['AB-42', 'C7', '-Q', '']

Expected output:
Positions: [2, -1, 0, -1]
```

Exercise 3 – Count and detect a status (2 points – full exam)

Partial I value: 6 points

A nested dictionary holds status strings for projects. Write one recursive function counting the exact status open and another checking if closed is present.

The two recursive functions examine leaf status strings in the nested dictionary, not dictionary keys. One returns the number of exact 'open' statuses; the other reports whether at least one exact 'closed' status occurs. Each function returns its own result, and empty dictionaries contribute no matches.

Input and assumptions

The outer object is a dictionary. Each value is either a string or another nested dictionary. Keys are strings; empty dictionaries may occur. Compare status strings exactly; the two functions have independent return values.

Required results and operations

a) Define the two separate recursive functions `count_open(data)` and `has_closed(data)`, each returning its own result
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 = {'a': 'open', 'b': {'c': 'closed', 'd': 'open'}}

print('Open:', count_open(data))
print('Closed:', has_closed(data))

Expected output:
Open: 2
Closed: True
```

Exercise 4 – Find the largest participant total (4 points – full exam)

Partial II value: 8 points

A registration office records quantities for different participants in a CSV file. Each row gives a name and one quantity, so the same name can occur several times. The office wants to identify the participant with the *largest combined quantity*, not simply the row containing the largest single value.

First combine the quantities for each distinct name. Then choose the name with the highest total; if two or more names are tied, select the alphabetically first one. Print exactly one line using the label `Largest total`. The supplied `records.csv` file is valid and has at least one data row.

Input and assumptions A valid header row is present, quantities are integers and names can repeat. Assume at least one data row.

Combine quantities by name, then print Largest total: followed by the winning name and total. Resolve ties alphabetically.

Required results and operations

a) Read `records.csv` with the standard `csv` module, where each row has a `name` and a `quantity`
b) Calculate the sum of quantities for each name, including names that occur on several rows
c) Print `Largest total: name total` for the largest combined quantity; resolve ties alphabetically

Tools and restrictions: Use csv.DictReader and standard Python data structures; pandas is not permitted. No error handling is required.

Reference example

```
records.csv:
name,quantity
Alice,3
Bob,6
Alice,5
Charlie,2

Expected printed result:
Largest total: Alice 8
```

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

Partial II value: 8 points

A delivery coordinator supplies three integer vehicle capacities as command-line arguments. The program reports the total number of packages each vehicle could handle over four identical trips. Create a separate function for this calculation, and print the three resulting totals in input order.

Input and assumptions Run `main.py` with exactly three arguments, each intended to represent an integer. No interactive input and no files are used.

Required results and operations

a) Define `trip_capacity(capacity, trips)` in `helpers.py`, returning the capacity multiplied by the number of trips

b) In `main.py`, read and convert the three command-line arguments using `sys.argv` and call the imported function with four trips
c) Print one computed total per line, preserving argument order
d) If an argument cannot be converted to an integer, print exactly `Invalid integer` and do not print any totals

Tools and restrictions: Use `sys.argv`, imports and `try-except ValueError`. No file processing or external libraries.

Reference example

```
Example command: python main.py 3 5 2
Expected output:
12
20
8
```

Exercise 6 – Battery usage (2 points – full exam)

Partial II value: 4 points

A handheld device stores its battery level after usage.

The device stores a current battery percentage. Using power changes that stored value according to the specified rule, with zero as the lower bound. The status check uses the final stored percentage and the defined threshold.

Input and assumptions

Usage amounts are nonnegative integers; do not introduce exceptions. No exception handling or external data files are required.

Required results and operations

a) Define `Battery` with device name and battery level
b) Implement `.use(amount)` without allowing the level below zero, and `.low()` (strictly under 20)
c) Use the battery and print the stored level and status

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

Reference example

```
Example usage:
battery_radio = Battery('Radio', 32)
battery_radio.use(18)
print(battery_radio.level)
print(battery_radio.low())

Expected output:
14
True
```