Simulation 01

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 01

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

Exercise 1 – Keywords in visitor comments (4 points – full exam)

Partial I value: 7 points

A visitor desk compares a list of short comments with a list of keywords. For each keyword, determine how many comments contain it.

The output has two levels: one count for each keyword, and one overall total. The overall total is the sum of the individual keyword counts, not the number of different comments. For example, one comment containing both 'museum' and 'tour' contributes to both keyword counts. Include every input keyword, even if its count is zero.

Input and assumptions

All keywords are distinct nonempty strings. Matching is case-sensitive and uses substring membership; no word tokenization is required. Empty lists are allowed.

Required results and operations

a) Produce a dictionary mapping every keyword to the number of comments containing that keyword
b) Print the keyword counts in alphabetical order, followed by the total number of keyword-comment matches
c) Count a comment at most once for the same keyword, but allow a comment to match several keywords
d) Use two nested for loops to compare each keyword with the comments

Tools and restrictions Use standard Python collections, functions when requested, and two genuinely nested for loops. No files or third-party libraries are needed in this exercise.

Reference example

```
comments = ['museum closes early', 'early entry for groups', 'museum tour']
keywords = ['museum', 'early', 'tour']

Expected output:
early: 2
museum: 2
tour: 1
Total matches: 5
```

Exercise 2 – Separate and summarize values (4 points – full exam)

Partial I value: 7 points

A sensor records a sequence of numerical readings. For its report, readings at or above a specified threshold must be selected, while lower readings are excluded.

The selection function receives the original readings and returns only those meeting the threshold. The summary function then receives that selected list, not the complete original list. Show the selected list and the returned tuple as two separate outputs. If no reading qualifies, both summary values are zero.

Input and assumptions All values are numeric; either selected list may be empty.

When no values are selected, summarize returns (0, 0).

Required results and operations

a) Implement select_high(values, threshold), returning selected numbers in their original order
b) Implement summarize(values), returning (sum, mean) for its argument
c) Use both functions on the reference list with threshold 10
d) Do not modify the input list

Tools and restrictions: Use functions with parameters and return values together with loops or list operations from Lectures 1–10; no libraries or file processing.

Reference example

```
values = [3, 12, 5, 16, 8, 11]

Expected output:
Selected: [12, 16, 11]
Summary: (39, 13.0)
```

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

Partial I value: 6 points

A collection of exhibition labels is represented by nested lists. Count the labels and check whether a given label is present.

The first function counts all string entries, including duplicate labels in different groups. The second function answers only whether the requested label appears anywhere; it does not return a count or a position. Test both functions on the supplied structure, and let them work on shorter, deeper or empty structures too.

Input and assumptions

The outer object is a list; elements are either strings or other nested lists. Empty lists may occur. Count every string, including repetitions; separately return True if the exact target occurs anywhere.

Required results and operations

a) Define the two separate recursive functions `count_labels(data)` and `contains_label(data, target)`, each returning its own result
b) Process nested lists at every depth; empty lists 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 = ['map', ['ticket', 'map'], [], ['guide']]
target = 'guide'

print('Count:', count_labels(data))
print('Found:', contains_label(data, target))

Expected output:
Count: 4
Found: True
```

Exercise 4 – Long lines in a text file (4 points – full exam)

Partial II value: 8 points

A visitor office keeps short messages in a text file. The program must identify the longest message and count the total number of words.

Each nonempty line is one message, not one word. The longest message is the line with the greatest number of characters after removing its line ending; word counting instead concerns all messages and uses whitespace-separated words. Print only the specified error message if the file cannot be opened.

Input and assumptions

The supplied file contains at least one nonempty line. Separate words by whitespace; if several messages have the maximum length, choose the first. Existing file contents are valid.

Required results and operations

a) Read messages.txt, with one message per nonempty line
b) Print the longest message as text, followed by the number of words across all nonempty messages
c) If the file is missing, catch FileNotFoundError and print Input file not found instead of a report

Tools and restrictions Use ordinary text files, string operations and FileNotFoundError. Do not use pandas or external libraries.

Reference example

```
messages.txt:
Welcome to the garden
Free entry
Today is sunny

Expected output:
Longest: Welcome to the garden
Words: 9
```

Exercise 5 – Service charges from CSV records (4 points – full exam)

Partial II value: 8 points

A facility records service usage in `usage.csv`, with columns `name,units`. Each record is billed separately at a fixed rate of 2 per unit, even when names repeat. The imported function calculates the charges for all records, while the main script handles file input, output and printing.

Input and assumptions The source file is supplied with valid headers. Records must be processed in their original order. No additional validation is required beyond that described.

Required results and operations

a) Create `calculations.py` with `calculate_charges(records, rate)`, which loops over the list of `(name, units)` pairs and returns the complete list of `(name, charge)` pairs
b) In `main.py`, read `usage.csv` using the standard `csv` module, convert units to integers and call the imported function once
c) Write `charges.csv` with the header `name,charge`, preserving row order, and print each returned name and charge

Tools and restrictions: Use basic loops, functions, imports, standard file handling and the named standard-library modules. No external packages are needed.

Reference example

```
Input file:
name,units
Alice,3
Bob,2
Charlie,1

Expected printed result:
Alice: 6
Bob: 4
Charlie: 2
```

Exercise 6 – Plant pot watering (2 points – full exam)

Partial II value: 4 points

A gardening application tracks water added to one potted plant.

The object represents one plant, with its plant name and current water quantity stored separately. After the specified call, only the water quantity increases; print the updated attribute rather than a value computed independently of the object. Do not introduce capacity restrictions or error handling.

Input and assumptions

All values are nonnegative; capacity limits are outside this exercise. No exception handling or external data files are required.

Required results and operations

a) Create a class `PlantPot` whose initializer stores `plant` and `water`
b) Implement `.add_water(liters)` to increase the stored quantity
c) Create the indicated object, add water and print the final water quantity

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

Reference example

```
Example usage:
pot_basil = PlantPot('Basil', 2)
pot_basil.add_water(3)
print(pot_basil.water)

Expected output:
5
```