Simulation 06

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 06

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

Exercise 1 – Count distinct days per visitor (4 points – full exam)

Partial I value: 7 points

The campus garden stores attendance records as (name, day) pairs; the same person may have multiple records on the same day.

The input records are visits rather than unique dates. Two identical visits for the same person on the same day count as one distinct day, while different days count separately. Produce one result per person, displayed alphabetically, without altering the original record collection.

Input and assumptions Each record is a pair of strings; repeated identical records are possible.

Repeated visits on the same day count only once for each person.

Required results and operations

a) Group the visit dates by person
b) Count distinct days, not the number of attendance records
c) Print each person alphabetically with the number of different days
d) Use a dictionary and sets

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

Reference example

```
visits = [('Alice','Mon'),('Bob','Tue'),('Alice','Mon'),('Alice','Wed'),('Bob','Tue')]

Expected output:
Alice: 2
Bob: 1
```

Exercise 2 – Accumulate complete batches (4 points – full exam)

Partial I value: 7 points

A warehouse processes batches of parcel weights in the given order. Processing ends after a complete batch makes the overall total reach a threshold.

Batches are indivisible: once a batch is started, all its parcel weights contribute, even if the target is passed halfway through that batch. Return both how many complete batches were included and the total weight of those batches. Stop only between batches or at the end of the input.

Input and assumptions

All weights are nonnegative integers and target is positive. Empty batches are allowed. Once a batch starts it is processed completely.

Required results and operations

a) Implement process_batches(batches, target), returning (processed_batches, total_weight)
b) Use a while loop to process batches until the target is reached or no more batches remain
c) For each processed batch, use a for loop to add its parcel weights to the running total

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

Reference example

```
batches = [[2, 3], [1, 4, 2], [9]]
target = 10

Expected output:
Result: (2, 12)
```

Exercise 3 – Select text reports from nested folders (2 points – full exam)

Partial I value: 6 points

A dictionary describes folders and subfolders, with filenames stored as dictionary values. Collect the filenames ending in .txt.

The dictionary stores filenames at various levels. Collect every string filename ending in '.txt', wherever it is found, and return them as a list. Names in different groups are separate occurrences even when identical. An empty folder or a structure with no matches contributes nothing.

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. Append the matching filename values, not the dictionary keys; preserve insertion order.

Required results and operations

a) Define `text_reports(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 = {'a': 'log.txt', 'b': {'c': 'table.csv', 'd': 'notes.txt'}, 'e': 'image.png'}

print(text_reports(data))

Expected output:
['log.txt', 'notes.txt']
```

Exercise 4 – Counting registration rows in a CSV (4 points – full exam)

Partial II value: 8 points

A registration office keeps a CSV log of submissions. Each data row represents one submitted record and contains the participant's name and a numerical quantity. The office wants to know how many records each person submitted, *regardless of the value of the quantity field.*

Read `records.csv` with the standard `csv` module and count the rows belonging to each name. For example, if Alice appears on two rows, her count is 2 even if those rows carry quantities 3 and 6. Print one line per participant in alphabetical order. This is a record-frequency task, not a sum-of-quantities task.

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

Print the number of CSV data rows associated with each name, in alphabetical order. Do not sum the quantity field: each record contributes one.

Required results and operations

a) Read `records.csv`, which contains a header `name,quantity`, using the standard `csv` module

b) Count how many separate data rows are associated with each name; do not sum the `quantity` field
c) Print each name and its record count in alphabetical order

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,3
Alice,6
Charlie,2

Expected printed result:
Alice: 2
Bob: 1
Charlie: 1
```

Exercise 5 – Apply a fixed increment to command-line values (4 points – full exam)

Partial II value: 8 points

A school activity program accepts three integers when it is launched from the terminal. For each number, it must calculate an adjusted score by adding six points. A separate helper module performs that calculation for one number; the main script reads the three arguments and prints the three adjusted scores in their original order.

The three values come from the script's command-line arguments, not interactive prompts or a file. The helper adds the fixed increment to a single integer. The main program displays adjusted values in input order; if any argument is not an integer, display the specified error without printing any adjusted values.

Input and assumptions

Execute main.py from a terminal and supply exactly three command-line arguments after the script name. They represent integer values; no interactive input() calls are needed. The increment is the fixed integer specified in the task

Define and import the helper function from helpers.py. Apply it independently to the three input numbers and print the transformed numbers on separate lines, preserving the order in which arguments were supplied

Required results and operations

a) Define adjust(value, increment) in a separate helpers.py module; it returns value + increment
b) Import the function into main.py
c) Read three integer values from sys.argv[1:] (not with input())
d) Print the adjusted values, one per line
e) Use a fixed increment of 6
f) If an argument cannot be converted to int, catch ValueError, print Invalid integer and do not print adjusted values

Tools and errors: Use sys.argv and the standard Python import mechanism. If any argument is not a valid integer, print exactly Invalid integer without printing partial results or a traceback

Reference example

```
Example command: python main.py 4 7 2
Expected output:
10
13
8
```

Exercise 6 – Reading goal tracker (2 points – full exam)

Partial II value: 4 points

A reading challenge tracks pages read toward a fixed goal.

A reading goal stores its target and completed pages as instance state. The shared class attribute belongs to the class, while progress for each reading goal is independent. After adding the reading session, display completed pages, pages still needed (not below zero) and the required shared attribute.

Input and assumptions

All page counts are nonnegative integers. No exception handling or external data files are required.

Required results and operations

a) Define `ReadingGoal` with book, target and completed pages initially equal to zero; define the class attribute `unit = "pages"`
b) Provide `.read(pages)` and `.remaining()` reporting how many pages are still needed, never negative
c) Create the goal, record the reading session and print the updated completed and remaining pages; also print the class unit attribute

Tools and restrictions Use a class attribute, instance attributes, methods and instance creation from Lectures 18–19.

Reference example

```
Example usage:
goal_novel = ReadingGoal('Novel', 100)
goal_novel.read(42)
print(goal_novel.completed)
print(goal_novel.remaining())
print(ReadingGoal.unit)

Expected output:
42
58
pages
```