Simulation 16

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 16

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

Exercise 1 – Filtering payment notes (4 points – full exam)

Partial I value: 7 points

A small office stores brief payment notes entered by its staff. Some notes indicate a completed payment and others indicate a pending or unrelated transaction. A note is suitable for the completed-payments report when it contains the word `paid` and does not contain the word `pending`. Both conditions refer to substrings within the same note.

The report should be a new list of qualifying notes. Remove leading and trailing spaces from each selected note, but do not change the words or their order. Keep the original sequence of qualifying entries and leave the input list unchanged.

Input and assumptions

The input is a list of lowercase strings that may contain spaces at either end. The list may be empty. Substring comparisons are case-sensitive, and no word tokenization is required.

Required results and operations

a) Define `completed_notes(notes)`, returning a new list containing only entries with `paid` but without `pending`
b) Remove leading and trailing whitespace from each selected note
c) Print the list of selected notes in their original order; print an empty list if none qualifies

Tools and restrictions Use a simple loop, Boolean expressions, strings and lists. No dictionary of frequencies or nested loops is needed in this exercise.

Reference example

```
notes = [' paid invoice ', 'pending payment',
         'overdue', 'paid rent']

Expected output:
['paid invoice', 'paid rent']
```

Exercise 2 – Counts and totals in daily tables (4 points – full exam)

Partial I value: 7 points

A shop records daily quantities in a list of rows. For each day, report the total quantity and the number of entries above a minimum.

Treat each row as a separate day's readings. Return one row total and one count of values strictly greater than the minimum for each row. Both resulting lists must correspond position by position to the original rows; an empty row contributes zeros.

Input and assumptions

Rows may be empty; all quantities are nonnegative integers. No library functions are needed.

Required results and operations

a) Define daily_report(rows, minimum), returning two lists: one of daily totals and one of daily counts strictly above minimum
b) Use two nested for loops and preserve the order of rows
c) An empty row must contribute a total and count of zero
d) Print the two lists obtained from the reference data

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

```
rows = [[2, 7, 4], [], [9, 1]]
minimum = 4

Expected output:
Totals: [13, 0, 10]
Counts: [1, 0, 1]
```

Exercise 3 – Sum electricity usage (2 points – full exam)

Partial I value: 6 points

An electricity monitor stores usage amounts in a nested dictionary. Return the total of all numeric leaf values.

Sum every numeric leaf value found in the nested dictionary, regardless of how many dictionary levels contain it. The keys are labels and do not contribute to the total. Empty nested dictionaries contribute zero.

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. All leaf amounts are integers; dictionaries are grouping containers, not numbers.

Required results and operations

a) Define `total_usage(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 = {'first': {'a': 4, 'b': 6}, 'second': {'c': 3}}

print(total_usage(data))

Expected output:
13
```

Exercise 4 – Smallest and largest reading (4 points – full exam)

Partial II value: 8 points

An office stores integer measurements and needs a report showing the smallest, largest and difference between them.

The source contains integer measurements on separate lines. The three statistics are the minimum observed value, the maximum and their difference (range). Write the same labeled report to the output file and to the screen.

Input and assumptions The file exists, all nonempty lines contain valid integers, and there is at least one integer.

Write summary.txt and print three labeled lines: Minimum, Maximum, Range, where Range is maximum minus minimum.

Required results and operations

a) Read all integers from values.txt, one integer on each nonempty line
b) Calculate the three requested summary values
c) Save the report in summary.txt and print the same lines

Tools and restrictions: Use ordinary text-file operations and Python numeric comparisons. No pandas, NumPy or exception handling.

Reference example

```
values.txt:
3
10
5
9
8

Expected printed result:
Minimum: 3
Maximum: 10
Range: 7
```

Exercise 5 – Adjusting attendance records (4 points – full exam)

Partial II value: 8 points

A CSV register contains an integer attendance count for every participant. Some entries may contain invalid text and must be skipped. The imported helper adjusts all valid participant records; the main program handles reading, validity checks and displaying the final summary.

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 `helpers.py` containing `add_bonuses(records)`, which accepts a list of `(name, count)` pairs and returns a complete list with 3 added to each count
b) Read `records.csv` in `main.py` and skip non-integer counts with `ValueError`
c) Call the helper once, print every resulting name and count and then print the total of adjusted counts

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

Reference example

```
records.csv:
name,count
Ada,4
Bob,invalid
```

```
Cara,2

Expected output:
Ada 7
Cara 5
Total: 12
```

Exercise 6 – Activity timer (2 points – full exam)

Partial II value: 4 points

An activity timer tracks minutes elapsed.

The timer stores elapsed time as object state. Advancing adds to the stored minutes; resetting restores the initial state. Show the value both after advancing and after resetting to demonstrate that the two operations affect the same instance.

Input and assumptions

The elapsed times are nonnegative integers. No exception handling or external data files are required.

Required results and operations

a) Define `ActivityTimer` with activity and an initial elapsed time of zero
b) Implement `.advance(minutes)` and `.reset()`
c) Create an object, advance the timer, print its value, reset it and print again

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

Reference example

```
Example usage:
timer_study = ActivityTimer('Study')
timer_study.advance(25)
print(timer_study.minutes)
timer_study.reset()
print(timer_study.minutes)

Expected output:
25
0
```