Simulation 05

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 05

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

Exercise 1 – Selecting valid attendance labels (4 points – full exam)

Partial I value: 7 points

A training centre records attendance using short labels. Labels beginning with `IN-` indicate a recorded arrival; the suffix `.ok` means that the arrival has been confirmed. Other labels may represent departures or unconfirmed entries and must not be included in the arrival report.

Produce a new list containing the confirmed arrival labels, in the same order as the input. To make the selected labels easier to read, remove only their leading `IN-` marker, leaving the rest of each filename-like label unchanged. For example, `IN-A.ok` becomes `A.ok`. The original list must not change.

Input and assumptions

The input is a list of strings, possibly empty. A label is accepted only when both its prefix and suffix match exactly; all comparisons are case-sensitive.

Required results and operations

a) Define `confirmed_arrivals(labels)`, returning a new list of the confirmed arrival labels with their `IN-` prefixes removed
b) Accept an item only if it both starts with `IN-` and ends with `.ok`; labels that meet only one condition are excluded
c) Preserve the input order, avoid modifying the input list and print the resulting list

Tools and restrictions Use one ordinary function, a list, a loop, Boolean conditions and string methods; no dictionaries, files or external libraries are needed.

Reference example

```
labels = ['IN-A.ok', 'OUT-B.ok', 'IN-C.tmp', 'IN-D.ok']

Expected output:
['A.ok', 'D.ok']
```

Exercise 2 – Averages in a dictionary of groups (4 points – full exam)

Partial I value: 7 points

A school stores the scores of several study groups in a dictionary. Each key is a group name, and its value is a list of scores. Produce a summary of the groups and identify the group with the highest average.

Each group can contain a different number of scores. Return one average per group and then identify the group whose computed average is greatest. The chosen group's name is the final result, not the maximum individual score. Follow the stated convention for empty groups and tied averages.

Input and assumptions

The dictionary is nonempty. Each key is a study-group name and each value is its list of numerical scores, which may be empty. When a group has no scores, use an average of zero. If the largest average is shared by several groups, choose the alphabetically first group. Do not modify the input dictionary or any of its lists.

a) Define `group_averages(groups)`, returning a new dictionary that associates each group name with its average score
b) Compute each average from the individual scores using loops; do not use `sum()` or numerical libraries
c) Define `best_group(averages)`, returning the name of the group with the highest average
d) Print the group averages in alphabetical order, then the name of the best group

Tools and scope Use ordinary functions, dictionaries and loops, including nested loops to process the groups and their scores. Only topics from Lectures 1–10 are needed.

Reference example

```
groups = {'Math': [12, 15, 18], 'Art': [],
          'History': [20], 'Physics': [10, 14]}

Expected output:
Art: 0
History: 20.0
Math: 15.0
Physics: 12.0
Best: History
```

Exercise 3 – Find the first matching city (2 points – full exam)

Partial I value: 6 points

A travel itinerary is a nested list of city names. Find the first city beginning with a given prefix.

The search result is a city name, not its index and not a count. 'First' means first in the natural order of the nested lists, descending into an inner group before continuing to subsequent elements. Return None if no city begins with the prefix. The comparison uses the exact supplied characters.

Input and assumptions

The outer object is a list; elements are either strings or other nested lists. Empty lists may occur. First means visiting elements from left to right and completing a nested sublist before the next element; return None if absent.

Required results and operations

a) Define `first_prefix(data, prefix)` and make it return the result described above for the entire structure
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 = ['Porto', ['Paris', 'Prague'], ['Rome']]
prefix = 'Pr'

print(first_prefix(data, prefix))

Expected output:
Prague
```

Exercise 4 – Find the most stocked item (4 points – full exam)

Partial II value: 8 points

A store records its current stock in `notes.txt`. Every line contains an item name and the quantity recorded in one stock entry; an item may be mentioned on several lines because its stock was recorded in separate batches.

The task is to identify the item with the *largest total quantity*. Combine all entries for the same item first, then compare the resulting item totals. If two or more items tie, select the alphabetically first name. Print only that winning item and its combined quantity using the label `Largest stock`. If the source file is missing, print `Input file not found` instead of the report. Names have no spaces and quantities are valid nonnegative integers.

Input and assumptions Each line contains one item name without spaces and one nonnegative integer; an item can appear repeatedly. Assume at least one nonempty record.

Print the name and combined quantity of the item with the greatest stock, using the label Largest stock:. If totals tie, choose the alphabetically first item.

Required results and operations

a) Read `notes.txt`, which contains an item name and integer quantity on each nonempty line
b) Calculate the total stock for each distinct item by combining its entries
c) Print `Largest stock: name total` for the item with the greatest total; on ties choose the alphabetically first name
d) If the file is absent, catch `FileNotFoundError` and print `Input file not found`

Tools and restrictions: Use ordinary text-file reading and a dictionary, not pandas. Only a missing input file requires exception handling.

Reference example

```
notes.txt:
paper 2
pens 5
folders 3
paper 2

Expected printed result:
Largest stock: pens 5
```

Exercise 5 – Inventory replenishment from CSV records (4 points – full exam)

Partial II value: 8 points

A community tool library compares the units requested for each tool with the units currently available. Each CSV row represents one independent request, even when the same tool appears more than once. The imported function processes all records and returns the transformed results. The main script handles CSV input and output, and displays the results.

Input and assumptions All input quantities are valid nonnegative integers. Repeated identifiers are allowed and must be processed separately. Preserve original row order.

Required results and operations

a) Create `inventory.py` containing `calculate_shortfalls(records)`, which iterates over all input records and returns a list of transformed pairs

b) In `main.py`, read `requests.csv` with the standard `csv` module, converting the numeric fields to integers, and call the imported function once
c) For each record, calculate how many additional units are needed: the larger of zero and the requested amount minus the available amount
d) Write `shortfalls.csv` with header `tool,missing` and print the returned values in the reference format

Tools and restrictions: Use ordinary functions, loops, modules, CSV reading and writing. Do not use pandas or external libraries.

Reference example

```
requests.csv:
tool,requested,available
Drill,7,4
Saw,2,5
Brush,6,1

shortfalls.csv:
tool,missing
Drill,3
Saw,0
Brush,5

Expected printed result:
Drill: 3
Saw: 0
Brush: 5
```

Exercise 6 – Room humidity meter (2 points – full exam)

Partial II value: 4 points

A room monitor stores a humidity percentage and detects overly humid conditions.

The humidity value belongs to the instance and changes when its update method is called. The condition method reports whether the updated percentage is strictly greater than 60, so exactly 60 is not considered high. Print both the current humidity and the Boolean result from that object.

Input and assumptions

The demonstration values remain in the range 0–100. No exception handling or external data files are required.

Required results and operations

a) Create `HumidityMeter` with room and humidity attributes
b) Provide `.change(delta)` and `.too_humid()` (strictly above 60)
c) Demonstrate the two methods on a meter object and print the results

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

Reference example

```
Example usage:
meter_office = HumidityMeter('Office', 55)
meter_office.change(8)
print(meter_office.humidity)
print(meter_office.too_humid())

Expected output:
63
True
```