Simulation 15

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 15

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

Exercise 1 – Find compatible delivery combinations (4 points – full exam)

Partial I value: 7 points

A dispatcher may combine each package weight with each available capacity. A combination is acceptable when the capacity is at least the weight.

A valid pair consists of one weight from the first list and one capacity from the second. Consider every possible pairing, not just matching positions. Retain only pairs whose capacity can accommodate the weight, preserving the order of weights first and then capacities.

Input and assumptions

Weights and capacities are positive integers. Repeated values represent separate possibilities. Either input list may be empty.

Required results and operations

a) Build a list of (weight, capacity) pairs that are acceptable
b) Use two nested for loops to consider every weight-capacity combination
c) Preserve the input order of weights, and within each weight the input order of capacities
d) Print the list of pairs and its length

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

```
weights = [3, 7]
capacities = [4, 8]

Expected output:
Compatible pairs: [(3, 4), (3, 8), (7, 8)]
Count: 3
```

Exercise 2 – Simple coordinate distances (4 points – full exam)

Partial I value: 7 points

Two lists contain the coordinates of different points measured at two times.

For the two equal-length coordinate lists, compare values at corresponding positions only. The first function returns the per-position absolute differences; the second summarizes those computed differences, not the unmodified original lists.

Input and assumptions

The lists have equal lengths and contain numbers; empty lists are allowed. The program must work with other valid data, not only this reference example.

Required results and operations

a) Implement `differences(before, after)` returning the absolute change at each corresponding position
b) Implement `total_change(differences_list)` returning the sum of these changes
c) Call the second function on the result of the first and print both outputs

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

```
before = [1, 4, 9]
after = [3, 1, 10]

Expected output:
[2, 3, 1]
6
```

Exercise 3 – Collect words containing a fragment (2 points – full exam)

Partial I value: 6 points

Notes are represented by nested lists of words. Return all words containing a given fragment, in their original order.

Collect every string containing the given fragment, at any nesting depth. The returned result is a new flat list, not a Boolean and not a number. Preserve encounter order and duplicate matching strings when they occur in the data.

Input and assumptions

The outer object is a list; elements are either strings or other nested lists. Empty lists may occur. Substring matching is case-sensitive; preserve every match, including duplicates.

Required results and operations

a) Define `collect_fragment(data, fragment)` 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 = ['code', ['decoder', 'cat'], ['barcode']]
fragment = 'cod'

print(collect_fragment(data, fragment))

Expected output:
['code', 'decoder', 'barcode']
```

Exercise 4 – Summarizing different event categories (4 points – full exam)

Partial II value: 8 points

A JSON document lists events of different types, potentially repeated. Summarize how many distinct event types appear and the total number of events.

The JSON event list may contain repeated categories. Combine the numeric counts for each category before computing the requested overall summary. Distinguish the number of distinct category names from the sum of all event counts.

Input and assumptions The JSON file has an events list; every event has a type string and an integer count. Event types may repeat. Assume at least one record.

Print the number of distinct event types and the sum of all count fields on separate labeled lines.

Required results and operations

a) Load the events list from events.json using the json module
b) Combine event counts by the type field
c) Display the required summary; handle FileNotFoundError with the exact message Input file not found

Tools and restrictions: Use the standard-library json module and dictionaries, not pandas. No JSON-format validation is needed.

Reference example

```
events.json:
{"events": [{"type": "walk", "count": 4}, {"type": "talk", "count": 5}, {"type": "walk", "count": 5}]}

Expected printed result:
Categories: 2
Events: 14
```

Exercise 5 – Department totals with pandas (4 points – full exam)

Partial II value: 8 points

An education centre records registrations for different courses in a CSV file. Prepare a report that combines registrations by course and adds a fixed allowance of two seats to each course total. The imported function must process the complete dictionary of category totals; the main script manages pandas input and writes the report.

Input and assumptions The supplied `enrolments.csv` has `course` and `students` columns with valid integer counts. Courses may appear in multiple rows.

Required results and operations

a) Define `add_allowance(totals, allowance)` in `helpers.py`. It receives a dictionary mapping courses to summed registrations and returns a new dictionary with the allowance added to each total
b) In `main.py`, read `enrolments.csv` using `pandas.read_csv()`, calculate the total registrations for each course using DataFrame selection and `.sum()`, and call the imported helper once with allowance 2
c) Print the adjusted course totals alphabetically by course and save them to `capacity.csv` with columns `course,capacity`

Tools and restrictions: This exercise requires pandas for CSV reading and DataFrame operations. No graphics or extra validation is required.

Reference example

```
enrolments.csv:
course,students
Python,3
Excel,2
Python,4
Excel,5

capacity.csv:
course,capacity
Excel,9
Python,9

Expected printed result:
Excel: 9
Python: 9
```

Exercise 6 – Parcel shipment weight (2 points – full exam)

Partial II value: 4 points

A parcel stores the mass of its contents.

The parcel stores one current weight. Adding an item increases the instance's weight; the fee calculation uses the updated value and the fixed rate given in the statement. Print the required values from the same parcel object.

Input and assumptions

Weights are positive numbers; the formula is fixed and needs no tariff lookups. No exception handling or external data files are required.

Required results and operations

a) Create `ShippingBox` with label and weight in kilograms
b) Define `.add_item(kg)` and `.shipping_fee()` returning `4 + 2 * weight`
c) Add one item and print the final weight and fee

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

Reference example

```
Example usage:
box_blue = ShippingBox('Blue', 2)
box_blue.add_item(3)
print(box_blue.kg)
print(box_blue.shipping_fee())

Expected output:
5
14
```