Simulation 04

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 04

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

Exercise 1 – Merging two inventories (4 points – full exam)

Partial I value: 7 points

The reading club receives two dictionaries containing quantities for the same types of items.

The two inventory dictionaries are separate records of quantities for the same possible items. Each distinct item appears once in the new merged dictionary. For shared items, the resulting quantity is the sum of both entries; for items appearing in one input only, keep their quantity. Print the combined report alphabetically.

Input and assumptions All quantities are nonnegative integers. Some keys may appear in only one dictionary.

A shared item is listed once with its combined quantity.

Required results and operations

a) Create a new dictionary containing every item appearing in either input dictionary
b) Sum quantities when an item appears in both
c) Print the final quantities in alphabetical item order
d) Leave both input dictionaries unchanged

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

Reference example

```
first = {'pen': 3, 'paper': 5}; second = {'paper': 2, 'box': 4}

Expected output:
box: 4
paper: 7
pen: 3
```

Exercise 2 – First qualifying reading in each group (4 points – full exam)

Partial I value: 7 points

A quality-control office stores several lists of readings. For each list, find the first reading that reaches a required minimum.

Each inner list is a separate set of readings. For each set, return its first value meeting the minimum, rather than its index or the number of qualifying values. Where no reading qualifies, return None at that group's position. The result must contain exactly one element per input group.

Input and assumptions

The input is a list of numeric lists. Groups may be empty. Preserve the order of the groups.

Required results and operations

a) Implement first_qualifying(groups, minimum), returning a list containing the first qualifying reading of each group
b) Put None in the result for a group without a qualifying reading
c) Use a for loop over the groups and a while loop to search within each group; stop scanning a group as soon as a match is found

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

Reference example

```
groups = [[2, 5, 9], [1, 3], [], [7, 11]]
minimum = 7

Expected output:
First values: [9, None, None, 7]
```

Exercise 3 – Total positive values in a nested dictionary (2 points – full exam)

Partial I value: 6 points

A project budget is kept in nested dictionaries. Find the sum of strictly positive integer amounts, ignoring zero and negative values.

Add only strictly positive numeric leaf values found anywhere in the dictionary hierarchy. Negative values and zeros do not contribute, and dictionary keys are never amounts. An empty nested group contributes zero. Return one number representing the total over the whole structure.

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. The grouping keys have no numeric meaning; an empty dictionary contributes zero.

Required results and operations

a) Define `sum_positive(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': 10, 'b': {'x': -4, 'y': 7}, 'c': {'z': 0}}

print(sum_positive(data))

Expected output:
17
```

Exercise 4 – Measurement summary from a text file (4 points – full exam)

Partial II value: 8 points

An inspection office stores one integer measurement per line in `values.txt`. It needs a short written report summarizing the measurements rather than copying the original lines. The report contains three quantities: the number of measurements, their total sum, and the largest individual measurement.

Read all the values from the text file, calculate the three results and write them to a new text file `summary.txt` using the labels `Count`, `Total` and `Maximum`. The supplied reference file contains five valid measurements. Assume the input contains at least one integer, so a maximum always exists.

Input and assumptions Use the supplied values.txt file. It contains one valid integer on each nonempty line; the measurements may be positive, zero or negative. No number should be hard-coded into the program

Determine the number of measurements, their sum and their largest value. Create summary.txt with exactly three labeled lines, Count, Total and Maximum, in that order. Use the values read from the file, not only the numbers from the example

Required results and operations

a) Read the integer observations in `values.txt`, with one value per line
b) Calculate the number of values, their total sum and their maximum
c) Write these results to `summary.txt`, one per line, with labels `Count`, `Total`, and `Maximum`

Tools and restrictions: Use the built-in file operations and numerical operations covered in class, without pandas or NumPy. Assume at least one measurement and a readable, correctly formatted input file; no extra input validation is required

Reference example

```
Input file (values.txt):
6
10
5
9
8

Expected result:
Count: 5
Total: 38
Maximum: 10
```

Exercise 5 – Batch calculation of squared readings (4 points – full exam)

Partial II value: 8 points

A text file contains one intended integer per line, although some lines may not represent valid integers. The main program must read and validate the input lines and then pass the entire list of valid numbers to an imported function. The helper transforms the list and returns the ordered results. The main program writes `squares.txt` and prints the requested total.

Input and assumptions The text file exists. Preserve the order of valid records and ignore blank lines or invalid integer values as specified.

Required results and operations

a) Create `helpers.py` defining `square_values(values)`, which loops over all numbers and returns a list of transformed values in the same order
b) In `main.py`, read `readings.txt`, skip invalid integer conversions by catching `ValueError` and call the imported function once
c) Write the returned list to `squares.txt`, one value per line, and print the sum as in the example

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

Reference example

```
readings.txt:
3
error
-2
4

Expected squares.txt:
9
4
16

Expected output:
Total: 29
```

Exercise 6 – Specialized musical instrument (2 points – full exam)

Partial II value: 4 points

A musical instrument has a name; a guitar is a more specific kind that also stores its string count.

The supplied base class already provides the general instrument description. The derived class adds the string count and adapts the description to include it, while retaining the inherited name. The test must create one derived instance and show the complete resulting description; do not rewrite the base class.

Input and assumptions

The `Instrument` class is provided as reference and is not to be changed. No exception handling or external data files are required.

Required results and operations

a) Define `Guitar` as a subclass of the provided `Instrument`; keep the parent class unchanged
b) Extend the child to store the number of strings, while reusing the parent initialization appropriately
c) Override `.describe()` to extend the inherited description with the additional information in the format shown
d) Create the reference instance and print its resulting description

Tools and restrictions Use basic single inheritance, method overriding and appropriate use of `super()` to reuse initialization and description behavior. No additional libraries or exceptions are required.

Reference example

```
Provided base class:
class Instrument:
    def __init__(self, name):
        self.name = name

    def describe(self):
        return 'Instrument: ' + self.name

Example usage:
guitar_acoustic = Guitar('Acoustic', 6)
print(guitar_acoustic.describe())

Expected output:
Instrument: Acoustic, strings: 6
```