Simulation 20

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 20

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

Exercise 1 – Unique visitor badges (4 points – full exam)

Partial I value: 7 points

An exhibition entrance records one badge code every time a visitor checks in. Some visitors may appear more than once in this sequence, and a separate set contains codes that have been blocked. The entrance needs a list of *distinct permitted badge codes* for its daily register.

An accepted code must not belong to the blocked set, and it must not have been added previously to the result. Preserve the order of the first permitted appearance: the register should contain each accepted code only once, even when the original input repeats it. This task concerns membership and uniqueness, not counting visits or adding numerical quantities.

Input and assumptions

The input is a list of badge-code strings and a set of blocked codes. Both may be empty. Codes are case-sensitive. The program does not interact with a real badge reader.

Required results and operations

a) Define `permitted_badges(entries, blocked)`, returning a new list of distinct codes not contained in `blocked`
b) If an accepted code occurs more than once, include it only at its first occurrence and preserve the relative order of accepted codes
c) Print the returned list; leave both inputs unchanged

Tools and restrictions Use lists, a set, membership tests, a loop and conditions. Do not use file operations or counting dictionaries.

Reference example

```
entries = ['A1', 'B2', 'A1', 'C3', 'B2', 'D4']
blocked = {'B2', 'Z9'}

Expected output:
['A1', 'C3', 'D4']
```

Exercise 2 – Review batches until enough samples (4 points – full exam)

Partial I value: 7 points

A technician analyzes groups of samples. Groups are processed in order, and analysis ends as soon as a full group brings the number of accepted samples to the required target.

Groups are accepted in order and each complete group contributes its qualifying samples to the count. Stop after the first processed group that makes the accepted count reach the target, or when no groups remain. Return both the number of groups processed and the final accepted count.

Input and assumptions

Groups are lists of nonnegative numbers, including possibly empty lists. target is positive.

Required results and operations

a) Implement accepted_batches(groups, minimum, target), returning (processed_groups, accepted_count)
b) Use a while loop for successive groups and a for loop to count accepted values within each processed group
c) A sample is accepted when its value is at least minimum; process every sample in a group before checking the stopping condition

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, 7, 9], [4, 8], [10, 1]]
minimum = 7
target = 3

Expected output:
Result: (2, 3)
```

Exercise 3 – Check and count CSV reports (2 points – full exam)

Partial I value: 6 points

A nested dictionary describes report filenames. Return the number of filenames ending in .csv and separately whether a particular filename appears.

The nested dictionary contains filename strings at its leaves. The first recursive function counts all names ending in '.csv', while the second checks whether one exact requested filename exists anywhere. Neither function counts dictionary keys as filenames; each handles empty dictionaries.

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. Filename matching is exact for the target; count every CSV filename, including repeats.

Required results and operations

a) Define the two separate recursive functions `count_csv(data)` and `contains_file(data, target)`, each returning its own result
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': 'report.csv', 'group': {'b': 'notes.txt', 'c': 'sales.csv'}}
target = 'sales.csv'

print('CSV:', count_csv(data))
print('Found:', contains_file(data, target))

Expected output:
CSV: 2
Found: True
```

Exercise 4 – Range of daily prices (4 points – full exam)

Partial II value: 8 points

A local market stores one integer price quotation per line in a text file. Prepare a report with the smallest quotation, the largest quotation, and their difference (the price range). Read `prices.txt` and write the result to `price_range.txt`.

Input and assumptions The input file exists and contains at least one valid integer. Values may be zero or positive. No extra validation is required.

Required results and operations

a) Read all integers in `prices.txt`
b) Calculate the minimum, maximum and difference between them
c) Write the three results to `price_range.txt`, with labels `Minimum`, `Maximum` and `Range`; print the same report

Tools and restrictions: Use ordinary text-file reading and writing and Python numerical operations. No external libraries.

Reference example

```
prices.txt:
7
10
3
10
8

price_range.txt and printed output:
Minimum: 3
Maximum: 10
Range: 7
```

Exercise 5 – Absolute-value report for measurements (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 `magnitudes.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 `absolute_values(values)`, which loops over all numbers and returns a list of transformed values in the same order
b) In `main.py`, read `source.txt`, skip invalid integer conversions by catching `ValueError` and call the imported function once
c) Write the returned list to `magnitudes.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

```
source.txt:
5
6
3

Expected printed result:
Total: 14
```

Exercise 6 – Remote worker description (2 points – full exam)

Partial II value: 4 points

A worker stores their name and role; a remote worker additionally has a working location.

The remote worker retains the ordinary worker's name and role but adds one location field. Its description builds on the parent's description and appends the location in the required format. Demonstrate by creating the derived instance and printing the result.

Input and assumptions

No employment rules or other validation are required. No exception handling or external data files are required.

Required results and operations

a) Define `RemoteWorker` as a subclass of the provided `Worker`; keep the parent class unchanged
b) Extend the child to store the working location, 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 Worker:
    def __init__(self, name, role):
        self.name = name
        self.role = role

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

Example usage:
worker_mila = RemoteWorker('Mila', 'Analyst', 'Porto')
print(worker_mila.describe())

Expected output:
Mila: Analyst (Porto)
```