Simulation 07

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 07

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

Exercise 1 – Names containing search fragments (4 points – full exam)

Partial I value: 7 points

An event organizer has a list of names and several text fragments to look for.

For each search fragment, count how many input names contain that fragment as a substring. A name can contribute to several different fragments but only once to each fragment. Show counts for all fragments, including those with zero matches, in alphabetical fragment order.

Input and assumptions

The fragments are distinct and nonempty. Matching is case-sensitive. A name can match more than one fragment.

Required results and operations

a) For each fragment, count how many names contain it as a substring
b) Store all counts in a dictionary and print them in alphabetical order of the fragments
c) Use two nested for loops so that every fragment is tested against every name

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

```
names = ['Marta', 'Martin', 'Paula', 'Marco']
fragments = ['Mar', 'a', 'tin']

Expected output:
Mar: 3
a: 4
tin: 1
```

Exercise 2 – Find the first out-of-range value (4 points – full exam)

Partial I value: 7 points

A temperature logger must identify the first reading outside its acceptable interval.

A value exactly on either boundary is acceptable. Return the index of the first value strictly below the lower boundary or strictly above the upper one; do not return the reading itself. If no value violates the interval, return None. Empty input also has no violating position.

Input and assumptions

All readings are numeric; low is at most high. The input may be empty.

Required results and operations

a) Implement first_outside(readings, low, high), returning the index of the first reading outside the inclusive interval
b) Return None if all readings are within the interval
c) Use a while loop that stops as soon as a reading outside the interval 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

```
readings = [18, 20, 22, 27, 19]
low = 17
high = 25

Expected output:
First outside: 3
```

Exercise 3 – Two checks on a number sequence (2 points – full exam)

Partial I value: 6 points

A sequence of integers needs two independent recursive checks: how many are even and whether any value is negative.

The two recursive functions answer different questions about the same list. One counts all even numbers, including repeated occurrences; the other reports whether at least one negative number exists. Each function must handle an empty list and return its result independently of the other.

Input and assumptions

The argument is a flat list of integers, not a nested list. Empty lists may occur. The list is flat and may be empty; define the two independent functions using recursion on progressively shorter lists.

Required results and operations

a) Define the two separate recursive functions `count_even(values)` and `has_negative(values)`, each returning its own result
b) Use recursion on the list rather than solving the task with a single loop; handle an empty list
c) Call the function(s) on the reference data and print the returned result(s) as shown

Tools and restrictions Use recursion with list indexing and slicing, not a loop that performs the entire computation. No additional libraries are required.

Reference example

```
values = [4, -3, 2, 7, 8]

print('Even:', count_even(values))
print('Negative:', has_negative(values))

Expected output:
Even: 3
Negative: True
```

Exercise 4 – Most popular event type from JSON (4 points – full exam)

Partial II value: 8 points

An events office receives a JSON file containing a list of event records. Each record specifies a type of event and the number of participants registered for it. A type can appear in several records, representing separate sessions.

Determine which event type has the largest *combined* number of participants. Add the counts for repeated types, compare the resulting totals, and print the winning type and its total. If several types share the maximum, use the alphabetically first one. Print `Input file not found` if the JSON file is missing; otherwise assume it is well formed.

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 event type with the greatest accumulated count and that count. Resolve ties using alphabetical order.

Required results and operations

a) Load `events.json` using the standard `json` module and read the list under `events`
b) Add the `count` fields of records with the same event `type`
c) Print the type with the greatest combined count using `Most events: type total`; resolve ties alphabetically
d) If the file does not exist, catch `FileNotFoundError` and print `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": 3}, {"type": "walk", "count": 2}]}

Expected printed result:
Most events: walk 6
```

Exercise 5 – File Processing with a NumPy Array (4 points – full exam)

Partial II value: 8 points

A numerical file is processed by a separate helper function, using a small NumPy calculation.

Use the NumPy operations specified in the question for the numerical transformation, but keep file reading and writing in the standard Python modules indicated. Clarify in the output which values come from the input and which are computed. Do not create a graph or use pandas.

Input and assumptions Read the supplied measurements.txt file containing one valid integer per line. Assume the file exists and contains at least one value. The specified increment applies to every measurement, including negative values

Create helpers.py with increase(values, amount) and main.py for file input and output. Convert the measurements to a NumPy array, obtain all adjusted values using the imported function, save them as integers to updated.txt, one number per line, and print the total of the adjusted values

Required results and operations

a) Create helpers.py containing a function increase(values, amount) which returns values + amount
b) In main.py, read the integers from measurements.txt, one number per line
c) Convert the list into a NumPy array using np.array()
d) Import increase and add 2 to each value
e) Write the resulting integers to updated.txt, one per line, and print their total

Tools and restrictions: For this exercise, use NumPy explicitly for the array operation, together with standard Python file handling and a normal module import. pandas is not required. Preserve the original order of measurements; no plotting or extra input validation is expected

Reference example

```
measurements.txt:
4
7
2
5

Expected updated.txt:
6
9
4
7

Printed total: 26
```

Exercise 6 – Step counter (2 points – full exam)

Partial II value: 4 points

A fitness app stores the steps recorded during a day.

The counter is attached to one device instance. Calling the update method changes its stored reading and the other method evaluates the new value. Create an instance, perform the stated operation and display the requested state; do not calculate a separate answer without updating the object.

Input and assumptions

Amounts and target are nonnegative integers. No exception handling or external data files are required.

Required results and operations

a) Define `StepCounter` with day and an initial step count of zero
b) Implement `.record(amount)` and `.reached(target)` returning a Boolean
c) Record two walking sessions and display the current steps and target status

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

Reference example

```
Example usage:
steps_monday = StepCounter('Monday')
steps_monday.record(3500)
steps_monday.record(2200)
print(steps_monday.steps)
print(steps_monday.reached(5000))

Expected output:
5700
True
```