Simulation 09

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 09

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

Exercise 1 – Find occupied columns by row (4 points – full exam)

Partial I value: 7 points

A small storage area is represented by a list of rows, where zero means an empty cell and a positive integer means an occupied cell.

Each row represents one day and each column a seat or timeslot. For every row, report the zero-based positions that satisfy the occupancy criterion, then store those lists under their corresponding row identifiers. Include rows with no matches as empty lists.

Input and assumptions

The grid is a list of lists of nonnegative integers. Rows may have different lengths and may be empty.

Required results and operations

a) Create a dictionary mapping each zero-based row index to a list of its occupied column indexes
b) Use two nested for loops to inspect every position of the grid

c) Include empty rows with an empty list; keep the columns in ascending order
d) Print the dictionary and the total number of occupied cells

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

```
grid = [[0, 2, 0], [1, 0], [3, 4, 0]]

Expected output:
Occupied columns: {0: [1], 1: [0], 2: [0, 1]}
Count: 4
```

Exercise 2 – Extract initials from names (4 points – full exam)

Partial I value: 7 points

A program formats the names of registered participants.

The result contains one initial character per nonempty name, in original input order. State explicitly what happens to empty strings using the rule in the exercise. Do not combine the initials into one long string if the task requests a list.

Input and assumptions Every name is a nonempty string containing at least one word, with words separated by whitespace.

The output list has one initials string for every input name.

Required results and operations

a) Implement initials(name) returning the first letter of each word, joined together; use a while loop to process the words
b) Implement format_initials(names) applying this operation to each name
c) Keep the input order
d) Print the resulting list

Tools and restrictions: Use functions, strings, lists and a while loop as taught in Lectures 1–10; no libraries or file processing.

Reference example

```
names=['Alice Green','Bob Stone','Cara Smith']

Expected output:
Initials: ['AG', 'BS', 'CS']
```

Exercise 3 – Count uppercase letters recursively (2 points – full exam)

Partial I value: 6 points

A short identifier is a string. Count how many individual characters are uppercase letters, using a recursive function rather than an explicit loop.

The input is a single string, not a collection of nested containers. Count the uppercase characters by a recursive function examining the string. Empty text contributes zero; lowercase letters, digits and punctuation do not count as uppercase letters.

Input and assumptions

The argument is a Python string, possibly empty. An empty string contains zero uppercase letters; punctuation and digits are not uppercase.

Required results and operations

a) Define `uppercase_count(text)` and make it return the result described above for the entire structure
b) Use recursion on the string and handle the empty string correctly
c) Call the function(s) on the reference data and print the returned result(s) as shown

Tools and restrictions Use recursion with string indexing and slicing as taught in the course; no loops or additional libraries are required.

Reference example

```
text = 'AbC-7D'

print(uppercase_count(text))

Expected output:
3
```

Exercise 4 – Select filename extensions from a file (4 points – full exam)

Partial II value: 8 points

A backup list contains one filename per line. Report all files ending with .csv.

Each line of the source file is a filename, not a path to be opened. Select names with the specified extension and produce the requested report, preserving the source order. No directory traversal or opening of the referenced files is required.

Input and assumptions

Filename matching is case-sensitive; repeated filenames are separate entries. The file may be empty.

Required results and operations

a) Read filenames.txt, with one filename per nonempty line
b) Select only names ending with .csv and print them in alphabetical order
c) Print the number of selected filenames on the final line
d) If the file is missing, handle FileNotFoundError with Input file not found

Tools and restrictions Use ordinary text-file handling and .endswith(), not filesystem walking or the os module.

Reference example

```
filenames.txt:
report.csv
photo.png
marks.csv
notes.txt

Expected output:
marks.csv
report.csv
Count: 2
```

Exercise 5 – Volunteer bonuses from CSV records (4 points – full exam)

Partial II value: 8 points

A CSV file stores pairs of names and integer quantities. Process every record independently and preserve the original row order. The imported function transforms the complete collection. The main program reads the source, saves `bonuses.csv` and prints the returned results.

Input and assumptions The source file is supplied with valid headers. Records must be processed in their original order. No additional validation is required beyond that described.

Required results and operations

a) Define `bonus_records(records)` in `bonus.py`, taking the entire list of name/quantity pairs and returning the complete list of name/converted-quantity pairs
b) Read `usage.csv` in `main.py`, convert quantities to integers and call the imported function once
c) Write `bonuses.csv` with columns `name,bonus` and print one name and converted value per line

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

Reference example

```
usage.csv:
name,units
Alice,5
Bob,2
Charlie,1

Expected printed result:
Alice: 7
Bob: 3
Charlie: 1
```

Exercise 6 – Gift card transactions (2 points – full exam)

Partial II value: 4 points

A gift card stores its remaining credit after purchases.

The gift card tracks one current balance. Spending reduces that balance by the requested amount, subject to the exact condition in the statement, while adding funds increases it. Report the resulting object state rather than recomputing it outside the class.

Input and assumptions

Prices are positive numbers; no exceptions are requested. No exception handling or external data files are required.

Required results and operations

a) Create `GiftCard` with code and balance
b) Implement `.purchase(price)` returning `True` and deducting the price if affordable, otherwise returning `False` without changing the balance
c) Demonstrate a valid purchase and print success and remaining balance

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

Reference example

```
Example usage:
card_G1 = GiftCard('G1', 20)
print(card_G1.purchase(7))
print(card_G1.balance)

Expected output:
```

```
True
13
```