Data Structures

Solve the following exercises and upload your solutions to Moodle until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

Exercise 1 – Submission: `a3_ex1.py` 25 Points

Write a program that reads in a matrix specified by the number of rows and columns, and then prints this matrix. The matrix must be created as follows (see the example input and output below for how it must look like):

- The user can enter the number of rows (integer) and number of columns (integer). You can assume that both numbers are in the range $[1, 10]$.
- The matrix must be created with a nested list.
- All elements of the matrix must be set to 0, except for matching row and column indices where the elements must be set to 1 instead. More formally,

$$M_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

  where M_{ij} is the matrix element at row index i and column index j.

The printing must be done according to the following rules (the ␣ character below represents a space, you do not have to literally print this character; see the example input and output below for how it must look like):

- The matrix must have a header which is formatted as follows: 2 spaces (␣) followed by ␣j for each column, where j is the column index. Directly below, a "line" must be printed as follows: 2 spaces (␣) followed by as many double-hyphens `--` as there are columns.
- Afterward, each row must be printed. A row starts with the row information i|␣, where i is the row index, followed by ␣M_{ij}, where M_{ij} is the matrix element at row index i and column index j.

Example input and output (right-hand side with spaces ␣ shown explicitly):[1]

```
Number of rows: 3
Number of cols: 4
   0 1 2 3
  --------
0| 1 0 0 0
1| 0 1 0 0
2| 0 0 1 0
```

```
Number␣of␣rows:␣3
Number␣of␣cols:␣4
␣␣␣0␣1␣2␣3
␣␣--------
0|␣1␣0␣0␣0
1|␣0␣1␣0␣0
2|␣0␣0␣1␣0
```

[1] Green colored text indicates user input from the console.

Example input and output (right-hand side with spaces ␣ shown explicitly):

```
Number of rows: 2
Number of cols: 10
   0 1 2 3 4 5 6 7 8 9
  --------------------
0| 1 0 0 0 0 0 0 0 0 0
1| 0 1 0 0 0 0 0 0 0 0
```

```
Number␣of␣rows:␣2
Number␣of␣cols:␣10
␣␣␣0␣1␣2␣3␣4␣5␣6␣7␣8␣9
␣␣--------------------
0|␣1␣0␣0␣0␣0␣0␣0␣0␣0␣0
1|␣0␣1␣0␣0␣0␣0␣0␣0␣0␣0
```

Example input and output (right-hand side with spaces ␣ shown explicitly):

```
Number of rows: 1
Number of cols: 1
   0
  --
0| 1
```

```
Number␣of␣rows:␣1
Number␣of␣cols:␣1
␣␣␣0
␣␣--
0|␣1
```

Example input and output (right-hand side with spaces ␣ shown explicitly):

```
Number of rows: 10
Number of cols: 10
   0 1 2 3 4 5 6 7 8 9
  --------------------
0| 1 0 0 0 0 0 0 0 0 0
1| 0 1 0 0 0 0 0 0 0 0
2| 0 0 1 0 0 0 0 0 0 0
3| 0 0 0 1 0 0 0 0 0 0
4| 0 0 0 0 1 0 0 0 0 0
5| 0 0 0 0 0 1 0 0 0 0
6| 0 0 0 0 0 0 1 0 0 0
7| 0 0 0 0 0 0 0 1 0 0
8| 0 0 0 0 0 0 0 0 1 0
9| 0 0 0 0 0 0 0 0 0 1
```

```
Number␣of␣rows:␣10
Number␣of␣cols:␣10
␣␣␣0␣1␣2␣3␣4␣5␣6␣7␣8␣9
␣␣--------------------
0|␣1␣0␣0␣0␣0␣0␣0␣0␣0␣0
1|␣0␣1␣0␣0␣0␣0␣0␣0␣0␣0
2|␣0␣0␣1␣0␣0␣0␣0␣0␣0␣0
3|␣0␣0␣0␣1␣0␣0␣0␣0␣0␣0
4|␣0␣0␣0␣0␣1␣0␣0␣0␣0␣0
5|␣0␣0␣0␣0␣0␣1␣0␣0␣0␣0
6|␣0␣0␣0␣0␣0␣0␣1␣0␣0␣0
7|␣0␣0␣0␣0␣0␣0␣0␣1␣0␣0
8|␣0␣0␣0␣0␣0␣0␣0␣0␣1␣0
9|␣0␣0␣0␣0␣0␣0␣0␣0␣0␣1
```

Exercise 2 – Submission: `a3_ex2.py` 25 Points

Write a program that repeatedly reads in elements/strings from the user until `"x"` is entered. All elements must be stored in encounter order and printed afterward. Additionally, only the unique elements must be identified and then also printed as a sorted list (ascending sort order). See the example input and output below for how it must look like.

Example input and output:

```
Enter element or 'x' when done: x
all: []
unique (sorted): []
```

Example input and output:

```
Enter element or 'x' when done: hi
Enter element or 'x' when done: hello
Enter element or 'x' when done: hello
Enter element or 'x' when done: abc
Enter element or 'x' when done: test
Enter element or 'x' when done: python
Enter element or 'x' when done: hi
Enter element or 'x' when done: abc
Enter element or 'x' when done: x
all: ['hi', 'hello', 'hello', 'abc', 'test', 'python', 'hi', 'abc']
unique (sorted): ['abc', 'hello', 'hi', 'python', 'test']
```

Exercise 3 – Submission: `a3_ex3.py` 25 Points

Write a program where the user can enter a string that contains comma-separated elements. This string must be split into its parts based on the separating comma character. For each string part, you then want to compute a simple hash value according to the following specification:

- For each character in such a string part, get its ordinal value by calling `ord(char)`, where `char` is the current character.
- The hash value of the entire string part is the sum of all character ordinal values.

Store the string part and its calculated hash value in a dictionary. If the dictionary already contains the string part (i.e., the user input contains equal string parts), use an `assert` statement to verify that the stored hash value of the existing dictionary entry is equal to the one you computed again. Finally, print each string part and its corresponding hash value (see the example input and output below for how it must look like).

Example input and output:

```
Enter comma-separated elements: test
'test' -> 448
```

Example input and output (assertion is internally tested two times):

```
Enter comma-separated elements: test,test,test
'test' -> 448
```

Example input and output:

```
Enter comma-separated elements: a,b,ab,hello,holle,ba
'a' -> 97
'b' -> 98
'ab' -> 195
'hello' -> 532
'holle' -> 532
'ba' -> 195
```

Exercise 4 – Submission: `a3_ex4.py` 25 Points

Write a program where the user can enter a string that contains comma-separated elements. This string must be split into its parts based on the separating comma character. For each string part, determine whether it is an integer number using the string method `string_part.isdecimal()`. If so, convert the string part to an integer and store it in a list. Otherwise, store the non-number in another, separate list. Additionally, count the occurrences of each unique integer number using a dictionary where the keys are the integer numbers and the values how often they occurred. Finally, print the list of all identified integers (in encounter order), the dictionary with their occurrence counts and the list containing all other, non-number elements (see the example input and output below for how it must look like).

Example input and output:

```
Enter comma-separated elements: test
integers: []
counts: {}
rest: ['test']
```

Example input and output:

```
Enter comma-separated elements: 512
integers: [512]
counts: {512: 1}
rest: []
```

Example input and output:

```
Enter comma-separated elements: abc,1,1,12,hello,1,5,1,5
integers: [1, 1, 12, 1, 5, 1, 5]
counts: {1: 4, 12: 1, 5: 2}
rest: ['abc', 'hello']
```

Important Information!

Please try to *exactly match the output* given in the examples (naturally, the input can be different). We are running automated tests to aid in the correction and grading process, and deviations from the specified output lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

For example, if the exercise has an output of
`unique (sorted): XYZ`
(where `XYZ` is the print representation of some object), do not write
`Unique [sorted]: XYZ`
(Uppercase `U` and square brackets instead of round parentheses) or
`unique (sorted):XYZ`
(missing space after the colon).

Feel free to copy the output text from the assignment sheet, and then change it according to the exercise task.