# 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. Use the provided **unit tests** to check your scripts before submission (**see the slides Handing in Assignments on Moodle**). 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.

**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).

Note that in general, after a `:` there is always a fixed white space. Feel free to copy the output text from the assignment sheet, and then change it according to the exercise task.

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

Write a program that creates 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 'X' if the sum of the row and column indices is an even number, otherwise 'O'.

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: 3 spaces (␣) followed by ␣$j$ for each column, where $j$ is the column index. Directly below, a "line" must be printed as follows: 3 spaces (␣) followed by a hyphen and then 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 $j$. The matrix elements are separated by spaces. Note that at the end of the line there is another character `␣|`.

- The last line is printed in the same matter as the one below the header indices.

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 | X O X O |
1 | O X O X |
2 | X O X O |
   ---------
```

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

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

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

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

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 | X O X O X O X O X O |
1 | O X O X O X O X O X |
2 | X O X O X O X O X O |
3 | O X O X O X O X O X |
4 | X O X O X O X O X O |
5 | O X O X O X O X O X |
6 | X O X O X O X O X O |
7 | O X O X O X O X O X |
8 | X O X O X O X O X O |
9 | O X O X O X O X O X |
   ---------------------
```

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

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

## 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. In addition, print the first and second half of the words that were entered. If the number of those words is odd, the second half should have one more element than the first one. Finally, print the (alphabetically sorted) words that are common to both halves of the entered elements. 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:  []
First half:  []
Second half:  []
Sorted common words:  []

Enter element or 'x' when done: hello
Enter element or 'x' when done: this
Enter element or 'x' when done: is
Enter element or 'x' when done: a
Enter element or 'x' when done: test
Enter element or 'x' when done: x
All:  ['hello', 'this', 'is', 'a', 'test']
First half:  ['hello', 'this']
Second half:  ['is', 'a', 'test']
Sorted common words:  []
```

Example input and output:

```
Enter element or 'x' when done: to
Enter element or 'x' when done: be
Enter element or 'x' when done: or
Enter element or 'x' when done: not
Enter element or 'x' when done: to
Enter element or 'x' when done: be
Enter element or 'x' when done: x
All:  ['to', 'be', 'or', 'not', 'to', 'be']
First half:  ['to', 'be', 'or']
Second half:  ['not', 'to', 'be']
Sorted common words:  ['be', 'to']
```

## Exercise 3 – Submission: `a3_ex3.py` 35 Points

Write a program for the JKU Pizzeria. Set up a dictionary where you add several pizzas and their prices:

- margarita, 10 Euros
- provenzale, 12 Euros
- rusticana, 17 Euros
- funghi, 16 Euros
- cipolla, 13 Euros

Note that for all pizza names in this exercise, lower case letters are used. The user then enters the desired pizza. If the pizza is not one of the available pizzas, display an error message as shown below (`Pizza not available, please try again.`) and the user needs to try again. This is done until the user enters a valid name of a pizza. Then a message with the price is displayed (for example `You have selected pizza cipolla for 13 Euros.`). Also create a list with extra ingredients that can be ordered (egg, cheese, salami, pepperoni, garlic, ham), these are displayed and the user enters the desired extras, separated by a semicolon (';').

This string must be split into its parts based on the separating character. For each part, you then need to check whether this extra is actually available (because people might enter extras that are not in the list of extra ingredients). In this case, display the not available and the available extras and compute a total price. Each extra is 1.5 Euros on top of the pizza price. If all requested extras are available you just display a `All extras available and added` and compute the price If no extras are selected you display `No extras selected`. See the example input and output below for how it must look like. and you again proceed to the final price.

Example input and output:

```
Hello at the JKU Pizza Service!
Please select your pizza:
Pizza margarita: 10 Euros
Pizza provenzale: 12 Euros
Pizza rusticana: 17 Euros
Pizza funghi: 16 Euros
Pizza cipolla: 13 Euros
Enter name of pizza: rusticana
You have selected pizza rusticana for 17 Euros.
Which extras would you like? Please enter them separated by a ';'.
egg
cheese
salami
pepperoni
garlic
ham
egg;ham;cheese
All extras available and added.
Your total price is now 21.5 Euros.
Thank you for your order!
```

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

```
Hello at the JKU Pizza Service!
Please select your pizza:
Pizza margarita: 10 Euros
Pizza provenzale: 12 Euros
Pizza rusticana: 17 Euros
Pizza funghi: 16 Euros
Pizza cipolla: 13 Euros
Enter name of pizza: cipolla
You have selected pizza cipolla for 13 Euros.
Which extras would you like? Please enter them separated by a ';'.
egg
cheese
salami
pepperoni
garlic
ham
eggplant;ham;cheese;fish
Extras not available: eggplant, fish
Extras available and added: ham, cheese
Your total price is now 16.0 Euros.
Thank you for your order!
```

Example input and output:

```
Hello at the JKU Pizza Service!
Please select your pizza:
Pizza margarita: 10 Euros
Pizza provenzale: 12 Euros
Pizza rusticana: 17 Euros
Pizza funghi: 16 Euros
Pizza cipolla: 13 Euros
Enter name of pizza: della casa
Pizza not available, please try again.
Enter name of pizza: frutti di mare
Pizza not available, please try again.
Enter name of pizza: margarita
You have selected pizza margarita for 10 Euros.
Which extras would you like? Please enter them separated by a ';'.
egg
cheese
salami
pepperoni
garlic
ham
ham;cheese
All extras available and added.
Your total price is now 13.0 Euros.
Thank you for your order!
```

Example input and output:

```
Hello at the JKU Pizza Service!
Please select your pizza:
Pizza margarita: 10 Euros
Pizza provenzale: 12 Euros
Pizza rusticana: 17 Euros
Pizza funghi: 16 Euros
Pizza cipolla: 13 Euros
Enter name of pizza: funghi
You have selected pizza funghi for 16 Euros.
Which extras would you like? Please enter them separated by a ';'.
egg
cheese
salami
pepperoni
garlic
ham

No extras selected.
Your total price is now 16.0 Euros.
Thank you for your order!
```

## Exercise 4 – Submission: `a3_ex4.py` **15 Points**

Write a program where the user can enter a string that contains space-separated elements. This string must be split into its parts based on the separating character. For each string part, determine its length and create a dictionary that uses the lengths as keys and the number of words with that respective length as the value. Print the dictionary in the manner shown below (sorted by the ascending length of the words). The output on the console should additionally also feature the words that belong to the respective length group (you do not need to remove duplicate words). If nothing is entered upon the initial prompt, only one entry should exist in the dictionary: 1 word with length 0 (which is the empty string).

Example input and output:

```
Enter text (space separated): a horse a horse my kingdom for a horse
length 1: 3 words (a, a, a)
length 2: 1 word (my)
length 3: 1 word (for)
length 5: 3 words (horse, horse, horse)
length 7: 1 word (kingdom)
```

Example input and output:

```
Enter text (space separated):
length 0: 1 word ()
```