# Agenda:

17:45 Doors open

18:00 Start of the event, introduction of the community

18:15 Introduction to Algorithmic Thinking

18:30 - 19:05 - first exercises

19:10 - 19:40 big exercise

19:40 - 19:50 Gina

19:50-20:15 Feedback and networking

# Notes:

* Below are only example codes, if they have something else working, that’s fine.
* Make sure to explain the concepts introduced in each exercise.
* try to have always only one operation/function/question on one card.
* If you see you cannot fit in the time, do not worry, we might start the second exercise a bit later.

1st group exercises

## 1. Step by step operations, variables, input, output - **finish by 18:35**

Introduce:

* variables (keeping values like date, text, numbers)
* operations (+, -, join etc)
* Input and output functions

**COUNTING DAYS - create the algorithm that gets a date and counts how many days are left**

### input (date1)

### date2= now()

### time\_left = date2-date1

### Output(time\_left)

**JOINING STRINGS - create the algorithm that takes name and surname and outputs the full name together**

### input(name)

### input(surname)

### full\_name=name+” “ +surname

### output(full\_name)

## 2. IF condition - **finish by 18:42**

Introduce:

* if condition
* else

Example without any code:

Look out the window. If it is raining outside, put on your rain boots and raincoat. Then go outside.

**Guessing game (0 to 10):**

### secret\_number = 10

### guessed\_number = input(“Ask for a number”)

### If guessed\_number equals secret\_number:

### output(“Correct answer”)

### Else:

### output(“Incorrect number. Game ended”)

**\*Challenge**: If not correct, give hint whether the number is lower or higher.

## 3. While loop - **finish by 18:50**

Introduce:

* while loop

**Guessing game that allows multiple guesses:**

*1st version:*

### secret\_number = 10 guessed\_number = input(“Ask for a number”)

### While guessed\_number is not secret\_number:

### output(“Incorrect number. Try again”)

### guessed\_number = input(“Ask for a number”)

### output(“Correct answer”)

*2nd version:*

### secret\_number = 10 guessed\_number = 0

### While guessed\_number is not secret\_number:

### guessed\_number = input(“Ask for a number”)

### If guessed\_number equals secret\_number:

### Say “Correct answer”

### Else:

### Say “Incorrect number. Game ended”

## 4. List + for each loop - **finish by 18:58**

Introduce:

* list (indexing from 0)
* for each loop
* break

**Go through the participants list and check on which position is John Bean**

### input(participants\_list)

### position = 0 for each participant in participant\_list: if participant is “John Bean”: break; position = position + 1 output(position)

**Challenge:** what if there is no John Bean in the list? how to make it work also in such a situation?

## 5. Functions - **finish by 19:05**

**Explain why code would be modularized into functions, how to avoid repeating code:**

Functions for data analytics: function that would convert tons to kilograms, or kilometers to miles (e.g. for localisation of taxi app)

**guess your position game! - using two previous exercises, create a game, where you need to guess your position on the list.**

*Note: you’ll need to change a bit existing papers and name them as a functions.*

### input(name)

### true\_position = check\_position(name)

### guess\_game(true\_position)

### 

Second group exercise (~30 minutes)

# Finish by 19:40

**Building a recommender system that would create a list of 7 to-watch movies for next week based on your IMDB watchlist.**

*You give some freedom in this task, especially un find\_similar\_movie function :)*

Helpful questions:

What are user needs?

What kind of data would we need for recommender system?

Where the recommender system will take data from?

How recommender system will make decision based on provided data?

How user interface will look like?

### movie\_recommendation\_list = new List

### movies\_watched\_list = get\_movies\_watched(username)

### movies\_watched\_list.trim(7) //take only first 7 entries

### for each movie in movies\_watched\_list:

### recommendation = find\_similar\_movie(movie)

### movie\_recommendation\_list.add(recommendation)

### output(movie\_recommendation\_list)

### 

### find\_similar\_movie(movie\_name):

### // idea 1: go for movies that people who rated this movie high also rated high

### // idea 2: go for a movie that somebody who watched this movie watched too

### // idea 3: go for a movie that has the same genre, actors or director

**If someone does not like cooking example or recommender system, maybe a game of some sort:** <http://cdn.cs50.net/ap/1516/problems/0/2/0-2.html#step_2_read_this>

# List for the 4th exercise:

1. Anna Drums
2. Benjamin Like
3. John Johnson
4. Patric Swayze
5. Ada Lovelace
6. John Bean
7. Michelle Bama
8. Elise Second