# Exercise 03 - Documentation, Typing, and Click

## Deadline

**November 16, 2020 - 12:00/noon (UTC+1:00)**

## Submission Guidelines

1. Clone your repository to your local workstation/computer if you have not done so
2. Structure and work on your submission.
    - Create a directory (folder) for the exercise sheet, e.g. name the directory ”exercise03” for Exercise03.
    - Place all relevant files/scripts in this newly made directory
3. Commit your work to the git repository.
4. Create a git tag.
    - Tag name must be equivalent to ”SubmissionExerciseSheet03”.
    - Tag can be made via the command line or using the GitLab GUI

## Grading (10 pts):

| Task | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Points | 1 | 2 | 2(+1) | 1 | 1 | 3 |

## Using Previous Homework

This exercise requires you to build on the work done for Exercise02. If you did not do or finish Exercise02, or do not wish to use your work, you may use the provided NetworkX script in the sample_files folder.

### Task 1 *Add Typing* (1 pt)

- Using the script you created in Exercise02 (or the sample file), go back through all of the defined methods and add type hints. This means that every method should have an expected data type for every defined parameter and return value.

    - Example:

```
def square_my_number(input_number: int) -> int:
    return input_number * input_number

def print_my_name(name: str) -> None:
    print(name)
```

### Task 2 *Add Docstrings* (2 pts)

- Add docstrings to every method in your script you created in Exercise02 (or the sample file).
    - Docstrings should include
    - a sentence describing what the method does
    - expected data types for all parameters and the return value
    - You may use whichever docstring style you like, but it is recommended you use either NumPy or Google

### Task 3 *Using Click* (2 pts)

1. Modify your command line interface methods to use **Click** instead. (2 pts)
    - The CLI should still take take two arguments: an input path for the PPI interaction file and an output path for the generated image

    - For the input file, Click should be configured to first check if the file exists
    - The name of command should be `create`
    - Example of executable command: `python3 my_script.py create /path/to/ppis.csv /path/to/output.pdf`
2. **(BONUS)** Add help documentation to your CLI methods using `help=` (1 pt)

### Task 4 *Input Options* (1 pt)

Users often have different file formats that wish to parse. Your code should be able to accept multiple file formats and generate a graph image. Alter your methods, and add additional Click functionality, so that users can either input the `ppis.csv` file OR a node list and an edge list.

- Use Click Options to define what file types are being passed:
    - `-p` / `--ppi` should be used if a PPI file is passed
    - `-n` / `--nodes` and `-e` / `--edges` should be used for passed node and edge lists respectively
    - Examples of executable commands:
    - `python3 myscript.py create -p ppis.csv /path/for/output.pdf`
    - `python3 myscript.py create -n nodes.tsv -e edges.tsv /path/for/output.pdf`

### Task 5 *Output Options* (1 pt)

Images come in all shapes and sizes and users should have multiple options for their image output.

- Modify your methods so that you can output an image in any of the following formats: `pdf`, `svg`, `png`, `jpg`
- **NOTE**: Be sure you handle unsupported file formats i.e. if a user tries to output the image as `docx`, your code should handle the error and warn the user what went wrong and how to fix it

### Task 6 *Shortest Paths* (3 pts)

Now that you can easily generate a graph from different kinds of files, it would be very helpful to run some basic graph algorithms on it. Here you must create a method that can return the shortest path between two protein nodes given their HGNC symbols and make it available via the CLI.

1. *Find the Path* (1 pt)

    - Create a method that takes two HGNC symbols and returns **the shortest path** between them
    - The method should return the single shortest path when possible, if multiple paths are returned with the minimum distance, then return all of them to the user
    - The user should have the option to print the path. Make sure it prints in a way that the user can understand i.e. print the sequence of HGNC symbols and NOT the identifiers
    - *Hint*: Use 'CREBBP' and 'TRA2B' as test source/target nodes respectively

2. *Highlight the Path* (1 pt)

    - Either modify your previous method, or generate a new one, that creates a PDF of your network with the shortest path(s) highlighted
    - All involved nodes and edges should be highlighted i.e. be a different color so that it stands out from the background

3. *Add it to the CLI* (1 pt)

    - Create a new CLI method (using Click) that allows the user to use these new shortest path features
    - CLI command should be named `path`
    - It must take two HGNC symbols as input (e.g. `--source` / `--target`)
    - Has the option to input either a PPI file (e.g. `--ppi`) or a node list and edge list (e.g. `--nodes` / `--edges`)
    - Has the option to print the path as a sequence of HGNC symbols to the terminal (think boolean flags…)
    - Has the option to generate an image of the network with the highlighted path(s) to a specified directory (e.g. `--output`)
    - Example working command: `python my_script.py path --source CREBBP --target TRA2B --ppi /path/to/ppis.csv --output ~/Desktop/graph.png`