Python School
OOP and Design patterns
Exercises

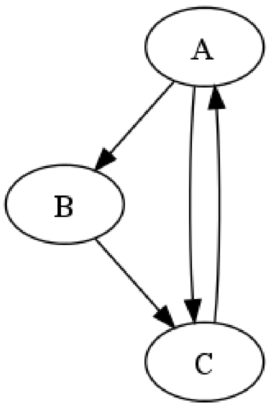
Authors: Bartosz Telenczuk, Niko Wilbert

1. (20 min) The `graph` module (provided in the repository) contains a set of classes for representing graphs. On a piece of paper reverse engineer its design:

   (a) Write down all class names, their methods and public properties; try to understand what all of them do (read the docstrings!).

   (b) Figure out how different classes are related (inheritance versus composition); draw a simple diagram.

   (c) Use the classes to represent the following graph:



2. (40 min) Modify the code in `starbuzz.py` to use the Decorator Pattern.

   (a) Define a class `BeverageDecorator` which is instantiated with a beverage object and contains two methods: `get_cost` which adds the cost of the decorator to the total drink cost and `get_description` which updates the description of the drink.

   (b) Subclassing `BeverageDecorator` define new ingredients: Milk and Cream. Use the ingridients to produce new drinks combinations.

   (c) (*Optional*) Write an unit test to test your code.

3. (30 min) Implement a Python iterator which iterates over string characters (ASCII only) returning their ASCII code (obtained by `ord` function):

   (a) Define a new iterator class which contains two methods:

   - `__init__` – a constructor taking the ASCII string as a argument,
   - `next` – returns the ASCII code of the next character or raises a `StopIteration` exception if the string end was encountered.

   (b) Define a new iterable class which wraps around a string and contains `__iter__` method which returns the iterator instance.

   (c) Test your code using explicit calls of `next` method (see example in the lecture) and for loop.

   (d) (*Optional*) Implement the same functionality using generators.

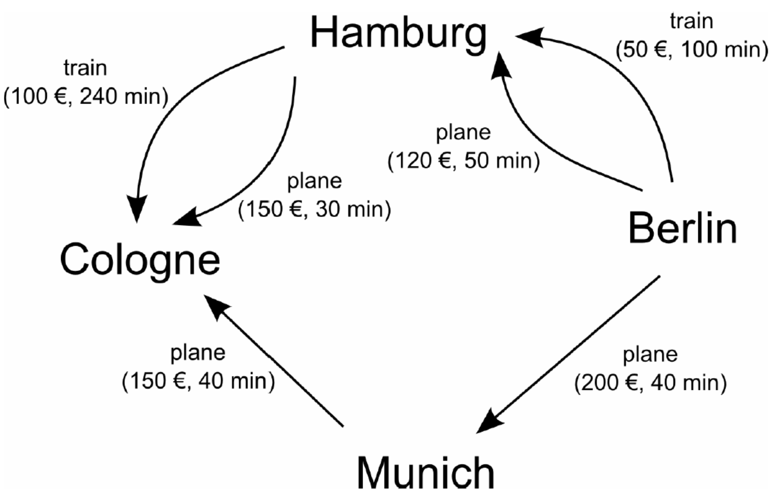
   (e) (*Optional*) Define a new iterator which returns the same ASCII codes but in a random order. Use it to iterate over your iterable object.

4. (55 min) Extend the `graph` library to solve a search problem. In this exercise, your goal is to write a travel planning application based on the `graph` module. We want to represent a set of cities as nodes in a graph, with edges between nodes representing different kinds of transportation.

   (a) Define a class `CityNode` which extends `Node` class by a new property `name` which is defined on class instantiation.

   (b) Define a class `TransporationEdge` extending `Edge` class. The edges should be directed and have two kinds of weights: travel `time` and `cost` and a short `description` defining the means of transportation.

   (c) Implement the following city graph as an example:



   (d) Now we want to find the quickest from Berlin to Cologne. Open `shortest_path.py` file. It contains `SearchAlgorithm` class, which implements Dijkastra algorithm for finding the shortest path in a graph.

   (e) Define a new class `SearchGraph` extending `Graph` class with methods for searching for the shortest path. Which design pattern can you use in the example?

   (f) Define new search algorithms to find the cheapest and fastest paths.

(g) Find the cheapest and fastest paths between Berlin and Cologne.