



**University of  
Zurich**<sup>UZH</sup>

**Department of Physics**



**Scientific Programming with Python**

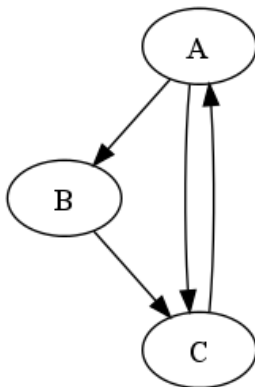
# OOP in Python Exercises

June 8, 2015

## Exercise 1: Understanding OOP (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:

- Write down all class names, their methods and public properties; try to understand what all of them do (read the docstrings!).
- Figure out how different classes are related (inheritance versus composition); draw a simple diagram.
- Use the classes to represent the following graph:



## Exercise 2: Decorator Pattern (30 min)

Modify the code in `starbuzz.py` to use the Decorator Pattern.

- 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.
- Subclassing `BeverageDecorator` define new ingredients: Milk and Cream. Use the ingredients to produce new drinks combinations.

### Exercise 3: Iterator Pattern (30 min)

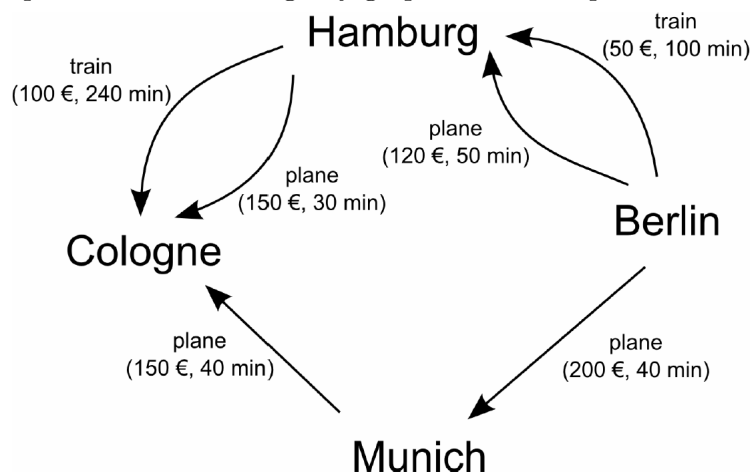
Implement a Python iterator which iterates over string characters (ASCII only) returning their ASCII code (obtained by `ord` function):

- (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.

### Exercise 4: Extending Classes (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 `TransportationEdge` 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 Dijkstra 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.

This exercise sheet is based on the exercises written by Bartosz Telenczuk, Niko Wilbert for the *Advanced Scientific Programming in Python School 2011*