# Object Oriented Programming in Python
## Exercises for class definition

1. Create an Employee class with the following data members and methods:
Data members: name, salary
Methods:
initializer (**__init__**(self)), toString (**__str__**(self)), getter and setter methods for data members, increasing salary with a given percentage
Create an EmployeeTest class in a separate package. Create an Employee object and use its methods.

Note: If data members are private (__name, __salary), we need to add getter / setter methods for accessing them. If data members are public (name, salary), we can access them directly from other classes.

2. Create **multiple initializers** for the Employee class. In a Python class there can be only one __init__ method. You can use default parameters in the __init__ method to simulate multiple initializations:

```
class Employee(object):
    def __init__(self, name, salary=150000):
        self.name = name
        self.salary = salary
```

When creating an object, we can use named parameters which also allows for writing multiple initializations.

```
employee1 = Employee(name = "Tom")
employee2 = Employee(salary = 250000, name = "Tom")
```

3. Add a birthday data member to the Employee class. Modify the initializers, add getter and setter methods. Write two methods with the same name (but with different input parameters) (**method overloading**). These methods create a date from different types of components. The first method creates a date from 3 integers. The second method creates a date from 2 integers and 1 string (month name).

4. Add an email **list as data member** to the Employee class.

5. Add a **static variable** (class variable), and a **static method** to the Employee class.
Static variable: age of retirement
Static method: check the age of employee, if he reached the age of retirement (@staticmethod)
Class method: create_date methods (see exercise 3) (@classmethod)

6. Add the position data member to the Employee class. Position must be one from a given list (**enum**).

7. **Operator overloading**:
Define __gt__ (> operator) in the Employee class. An Employee is greater than the other, if he is older / has higher position / has greater salary.

8. Create an **EmployeeArray** class. Its initializer takes 1 argument, the size of the array. Add some array algorithms to the class. For example: compute the average salary, or sort the employees by name.