Introduction to Object-Oriented Programming in Python

- Object-oriented programming (OOP) involves programming using objects.
- An object represents an entity important for an application; similar objects share the same structure (data/properties/variables) and behavior (functions/methods).
- Objects are defined using Class definitions. The concept of `Class` is synonymous with `type`.
- Each object is an instance of a class. Instance variables are tied to specific objects. Each object has its own instance variables.
- In addition, there are class variables, associated with the class and shared by all objects of that class.

A Python class uses variables to store data fields and defines methods to perform actions. Additionally, a class provides a special type method, `__init__()`, known as initializer, which is invoked to initialize the data components of a newly created object.

An initializer can perform any action, but initializer is designed to perform initializing actions, such as creating the instance variables (a.k.a. data fields) of an object.

**EXAMPLE**

Let’s create a class definition for object type Review. Each review should have a numeric rating, a maximum possible rating and review text. Since these values are going to be different for different Review objects, they define properties of each individual object, thus are called instance variables `rating`, `maxrating` and `reviewText`.

```python
'''Version 1 of the Review class; includes the __init__ method. Demonstrates instance variables and parameter 'self' - the obligatory reference to the calling object.'''

class Review:
    # Method __init__ is called automatically during class instantiation (a.k.a. object construction)
    def __init__(self, rat, rtext, outof = 5):
        ''' self refers to the newly created Review object '''
        self.maxrating = outof  # instance variable
        self.rating = rat       # instance variable
        self.reviewText = rtext # instance variable

def main():
    # Create two objects of type Review
    r1 = Review(4, 'Friendly reception very helpful. Hotel has pleasant views.' )
    r2 = Review(7, 'Nice staff. Nice location.', 10)

    # Access instance variables
    print ('r1 :', r1.rating, 'out of ', r1.maxrating, ' Text:', r1.reviewText)
    print ('r2 :', r2.rating, 'out of ', r2.maxrating, ' Text:', r2.reviewText)

    r2.rating = 55
    print('-- after an assignment ---

    print ('r1 :', r1.rating, 'out of ', r1.maxrating, ' Text:', r1.reviewText)
    print ('r2 :', r2.rating, 'out of ', r2.maxrating, ' Text:', r2.reviewText)

main()
```
OBJECT CONSTRUCTION

Once a class is defined, you can create objects from the class by using a constructor, e.g. Review(). Define method __init__(), which will be called automatically when the object is constructed. The same parameters that were passed to Review() constructor will be passed to the __init__(). After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the object member access operator.

self is a parameter that represents the constructed/ the calling object. Using self, you can access instance variables in an object. Instance variables are for storing object properties. You can use the syntax self.x to access the instance variable x for the object self in a method.

INSTANCE VARIABLES VS CLASS VARIABLES

Instance variables
- define properties of each individual object
- must be introduced by methods of a class (differently from Java)
- are typically introduced in the __init__ () method. (rating, maxrating and reviewText)
- inside class methods must be referenced with keyword self, which refers the the created/calling object itself.
- outside of class methods, referenced using the dotted notation, (e.g..r1.rating in main)

Class variables (corresponds to static in Java)
- define properties of the entire class type, e.g. MAX_TEXT_LENGTH
- must be introduced inside the class but outside of methods (differently from Java)
- one per class
- accessed using the name of class, e.g. Review.MAX_TEXT_LENGTH

INSTANCE METHODS

All instance methods defined inside a class, have self as the first parameter, which refers to the object that invokes the method. You can use any name for this parameter. But by convention, self is used.

```python
''' Version 1 of the Review class;
- adds a class variable, checking for validity in construction and an instance method printReviewData() '''
class Review:

    MAX_TEXT_LENGTH = 1000 #CLASS VARIABLE: max size of review text allowed

    def __init__(self, rat, rtext, outof = 5 ):
        ''' self refers to the newly created Review object '''
        self.maxrating = outof

        # Check the rating value is valid, before assigning
        if rat >= 1 and rat <= outof:
            self.rating = rat
        else :
            print(’setting rating to 1’)
            self.rating = 1

        # Check the text length
```
```python
if len(rtext) <= Review.MAX_TEXT_LENGTH:
    self.reviewText = rtext
else:
    self.reviewText = rtext[:Review.MAX_TEXT_LENGTH]

def printReviewData (self):
    '''Instance method. Self is the OBLIGATORY FIRST parameter, which refers to the calling object'''
    print('Rating is ', self.rating, 'out of ', self.maxrating, 'Text:', self.reviewText)

def main ():
    print ('Maximum allowed review length is ', Review.MAX_TEXT_LENGTH, 'characters')
    r1 = Review (4, 'Friendly reception very helpful. Hotel has pleasant views.' )
    r2 = Review (7, 'Nice staff. Nice location.', 10)
    print ('r1 :', end = '')
    r1.printReviewData ()
    print ('r2 :', end = '')
    r2.printReviewData()

    print('-- after an assignment ---')
    r2.rating = 55
    print ('r2 :', end = '')
    r2.printReviewData()

main()
```

Practice problem:
1. Add instance method normalizedRating() to the Review class, which will return an integer percentage value of the rating, i.e. what percentage of maxrating the rating value comprises. For example, r1. normalizedRating() must return 80.

PRIVATE INSTANCE DATA (INFORMATION HIDING)

To prevent direct modifications of data fields, don’t let the client directly access data fields. This is known as information hiding. This can be done by defining private instance variables. In Python, the private data fields are defined with two leading underscores. You can also define a private method named with two leading underscores.

To access (get value set value) of a private variable, define public getter and setter methods, e.g. getRating()/setRating().

- Getter functions have no input parameters and return the value of the appropriate instance variable.
- Setter functions take an input parameter, which is used as a new value for the instance variable. The setter method must check if the new value is valid for the object and in such case, assign that value to the instance variable.

-3-
''' Review class with PRIVATE class and instance variables, without getter/setter functions'''

```python
class Review:
    __MAX_TEXT_LENGTH = 1000  # maximum size of review text allowed

    def __init__(self, rat, rtext, outof = 5):
        ''' self refers to the newly created Review object '''
        self.__rating = outof

        # check the rating value is valid, before assigning
        if rat >= 1 and rat <= outof:
            self.__rating = rat
        else:
            print('setting rating to 1')
            self.__rating = 1

        # Check the text length
        if len(rtext) <= Review.__MAX_TEXT_LENGTH:
            self.__reviewText = rtext
        else:
            self.__reviewText = rtext[:Review.__MAX_TEXT_LENGTH]

    def printReviewData(self):
        print('Rating is ', self.__rating, ' out of ', self.__maxrating,\
              ' Text:', self.__reviewText)

def main():
    # print ('Maximum allowed review length is ', Review.__MAX_TEXT_LENGTH, ' characters')  # causes AttributeError:

    r1 = Review( 4, 'Friendly reception very helpful. Hotel has pleasant views.' )
    r2 = Review( 7, 'Nice staff. Nice location.' , 10)

    print ('r1 :', end='')
    r1.printReviewData ()
    print ('r2 :', end='')
    r2.printReviewData()

    # Following will cause AttributeError
    #print ('r1 :', r1.rating, ' out of ', r1.maxrating, ' Text:', r1.reviewText)
    #print ('r2 :', r2.rating, ' out of ', r2.maxrating, ' Text:', r2.reviewText)

    print('-- after an assignment ---')
    r2.__rating = 55  #### WATCH OUT - this creates a different variable, but does NOT affect the PRIVATE one
    print (r2.__rating)

    print ('r2 :', end='')
    r2.printReviewData()
```
```
Practice problem:
2. Add setters and getters (method that set and get values of instance variables, e.g. getRating()/setRating()) to the Review class.

**Design guidelines**: If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields private. If a class is only used internally by your own program, there is no need to encapsulate the data fields.

Practice problem:

3. Define class Hotel with the following **private instance data**: hotel **name** (string value) and a list of **reviews** (list of Review type objects).

You will need to import the Review class definition as follow (assuming it is in Review.py which does not have any executable code, only definitions):

```python
from Review import Review
```

- The __init__() method should set the name of the hotel and reviews list to the values passed as a parameters.
- Add methods:
  - instance method printHotelData() to output hotel information (name, reviews)
  - instance method addReview() which will add the review passed as a parameter to the list of reviews
  - method main() that creates a Hotel object and tests the developed methods.
  - Add functionality to printHotelData() to print the average normalized review score, as a percentage value, producing output that looks like the following:

```
Hotel "The Grand Budapest Hotel"
Rating is 5 out of 5 Text: Unique hotel. Makes its own pastries.
Rating is 8 out of 10 Text: Nice place. Great entertainment.
Rating is 12 out of 12 Text: Best hotel ever.
Average normalized rating is: 93%
```

- Add method mostFrequentWords() which will output the three most frequent words in the hotel reviews, excluding the stop words (see stopwords.py from Week 9) and words ‘hotel’, ‘hotels’, ‘room’ and ‘rooms’. Test the method by calling it in the main().