ENGR 101 - Introduction to Programming
Study Questions - Week 14

1. What is the output of the following code?

```
class Car:
   a = "s"

   def __init__(self, a):
      self.a = a

c = Car("sahin")
print c.a
```

2. Now, What is the output? What is the difference between the two fragments of code?

```
class Car:
   a = "coper"

   def __init__(self, a):
      self.a = self.a

c = Car("sahin")
print c.a
```

3. Write a class Person that will have attributes first name and last name. Write a method to get the full name. Now write another class Employee that will inherit from class Person and but you will add an id_number attribute to it. Create an object Employee and print their name using the method that you wrote in the parent class.
4. Write a class Car which will have max_speed as its attribute. Create two different classes for two different brand cars (e.g., Bmw and Audi) which will inherit from the Car class and write a method price which will return the price for the corresponding car brand.
5. Create a class Point_2D that will have as attributes an x and y coordinate, by default they should be zero. Now create a second class using inheritance to represent a point in 3D.

6. Create 3 classes, FirstName, SecondName and FullName. The first **class** will have a method called **"first_name"** that returns the name **"Mohamed"**. The second **class** will have a method called **"second_name"** that returns the name **"Ali"**. The third **class** will inherite the two classes **and** have a method called **"full name"** that prints the full name when called. You should call the two methods (first_name **and** second_name inside the **"full_name"** method).

7. What would be the output of this code:

```
class Student:
     def __init__(self,name,id):
          self.name = name
          self.id = id
class CurrentStudent(Student):
     def __init__(self,name, id, semester):
```

```
            Student.__init__(self, name, id)
            self.semester = semester

std1 = CurrentStudent('Ahmet','213456',4)
std2 = Student('Ayse','136242')

print isinstance(std1,Student)
print isinstance(std1,CurrentStudent)
print isinstance(std2,Student)
print isinstance(std2,CurrentStudent)
```

8. Write a class Game with attributes energy, money and no_of_castles, and a show_info (which will print the attributes' values) method. Write a class named Player that will inherit from Game class and implement a create_new_castle method which will allow the player to create a new castle if his energy is >5 and his money is > 10, it should update the attributes accordingly.

9. Create a class, Document, with a name and number of pages attribute, which will be used to inherit when creating two different classes named Word and Excel. Both of them should have a method that will display their content and another method that will display their number of pages. Create a list of document objects of type Excel and Word and then call the method to display their content and their number of pages.

10. Create another class OponentPlayer that will inherit from class Game written before and it will have an additional attribute named no_of_destructions. Implement a method that will destroyCastle() which will allow the player to destroy a castle, thereby increasing his no_of_destructions attribute. If the player's energy is lower than 10, he won't be able to destroy the castle.

11. Define a class Polygon that takes number of sides to initialize. It has an enterSides method that prompts user to enter all n sides of the polygon (e.g. user is prompted 3 times for a triangle). Another method, showSides, prints all sides, one per line.

12. Now define a class, Triangle, whose superclass is the Polygon class (i.e. Triangle inherits from Polygon). It has an additional method area that returns area of the triangle using Heron's formula given below.

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

Note: A is for area, s is perimeter of triangle divided by 2, and a, b, c are the sides.

13. Define a class Date that has three attributes: day, month, and year. This class inherits from the Time class you worked with in lectures. Add an __str__ method that prints something like 20161231 23:59:00.

14. Define a class Student that can be initialized using two attributes: name and surname. Add an __str__ method to it that prints person's name (e.g. Mehmet Kadir). Define a class CollegeStudent that inherits from Student class defined in Q3 and has three additional attributes: department, studentID, and year of enrollment. Also, add an __str__ method that prints these additional attributes as well (e.g. an example output would be: Mehmet Kadir, CS, 213039104, 2013)

15. Recall the RationalNum class from Q1. Overload all the following comparison operatos for it.

```
<       __lt__(self, other)
<=      __le__(self, other)
==      __eq__(self, other)
!=      __ne__(self, other)
>       __gt__(self, other)
>=      __ge__(self, other)
```

16. You can also practice operator overloading using different data structures in Python. Define a class PowerList that takes a list of integers (or floats) to initialize (you can access the list any time using the attribute name you assigned to it). Use concepts from type-based dispatch and overload the multiplication operator * by adding a __mul__ method that returns a modified version of the original list in which every element has been multiplied by the given integer n.

17. Now overload power operator ** in the same PowerList class by adding a __pow__ method. It returns a modified version of the original list in which every element has been raised to the given power.

18. Define a method __abs__ that will make the list work with built-in abs function (think polymorphism). This method will return a modified version of the original list in which every element has been replaced by its absolute value.

19. Define a class PowerDict that takes a dictionary to initialize (you can access the dictionary any time using the attribute name you assigned to it). Overload the addition and subtraction operators for it. The addition operation will yield a dictionary which will contain combinations of keys-valued from added dictionaries (union operation), and the subtraction operation will yield a dictionary which has elements that are unique to the dictionary on the left side of the operator. Use __rsub__ also, if needed.

20. What is the output of the code below?

```
class MyClass:
    def __init__(self, inp):
        self.xyz = inp

    def __str__(self):
        return str(self.xyz*2)

class MyClass2(MyClass):
    def __str__(self):
        return str(self.xyz*3)

test = MyClass2(range(1,3))
print test
```