PGR107 - Python Programming

Kristiania University College

Final Exam

Academic contact during examination: Hadi Zahmatkesh, hadi.zahmatkesh@oslomet.no, +47 93255316

Technical contact during examination: eksamen@kristiania.no

Exam type: Written home examination in groups (1-5 students)

Support materials: All support materials are allowed

Plagiarism control: We expect your own independent work. Any copies from the internet or other groups will give you an automatic **FAIL** for every involved party (the giver and the taker).

Grading scale: Norwegian grading system using the graded scale A - F where A is the best grade, E is the lowest pass grade and F is fail.

- **A**: Excellent and comprehensive understanding of the topics
- **B**: Very good understanding of the topics
- **C**: Good understanding of the topics
- **D**: Satisfactory understanding of the topics but with significant shortcoming
- **E**: Meets the minimum understanding of the topics
- **F**: Fails to meet the minimum academic criteria

Learning outcomes (knowledge & skills):

The student

- understands problem solving using programming.
- understands the principles of object-oriented programming.
- has overall knowledge of general properties of python programming language such as program flow, loops, and choices.
- has knowledge of Python programming using data structures, functions, classes, objects, and modules.

Submit complete Python codes (.py files) for Questions 1-5.

Question 1 (20 points)

In this question you are asked to use ***classes*** and ***inheritance***. (The exercise can be solved in several ways, however, use of ***inheritance*** gives full score).

We want to implement two classes (***Student*** and ***Employee***) that inherit attributes and methods of another class (***Person***).

The ***Person*** class will store information (**first name**, **last name**, and **age**) about a person, and it will have a method that prints this information. Here is the implementation of the ***Person*** class:

```
class Person:
    def __init__(self, fname, lname, age):
        self.fname = fname
        self.lname = lname
        self.age = age

    def get_info (self):
        print ("Full Name:", self.fname, self.lname)
        print ("Age:", self.age)
```

The ***Student*** class has the same attributes as the ***Person*** class (**first name**, **last name**, and **age**). It also has another attribute (**student ID**) that is specific to students. Note that you have to initialize any similar attributes using the ***Person*** class __init__ method and make them available in the ***Student*** class. In addition, the ***Student*** class has a method **get_stuinfo ()** that prints the student's full name, age, and student ID. Note that you need to use the **get_info ()** method in the ***Person*** class to print the full name and age.

The ***Employee*** class has the same attributes as the ***Person*** class (**first name**, **last name**, and **age**). It also has two more attributes (**employee number**, and **salary**) that are specific to employees. Note that you have to initialize any similar attributes using the ***Person*** class __init__ method and make them available in the ***Employee*** class. In addition, the ***Employee*** class has a method **get_empinfo ()** that prints the employee's full name, age, employee number, and salary. Note that you need to use the **get_info ()** method in the ***Person*** class to print the **full name** and **age**.

Use the above description to implement these two classes: ***Student*** and ***Employee***.

The code below should work with the expected output if your implementation is correct.

```
new_student = Student ("Anthony", "Smith", 35, "s346571")
new_student.get_stuinfo ()
print ("==========================")
new_employee = Employee ("Sarah", "Tayolr", 34, 2919736, 5000)
new_employee.get_empinfo ()
```

Output

```
Full Name: Anthony Smith
Age: 35
Student ID: s346571
====================
Full Name: Sarah Tayolr
Age: 34
Employee No: 2919736
Salary:  5000 USD
```

Question 2 (20 points)

a. Write a class called ***Student*** that has two instance variables, ***name*** and ***mark***, and one class variable, ***passingMark***. Assign 50 to passingMark. Within the class, you need to code the **__init__** and **__str__** methods. The **__str__** method returns the name and mark of the Student's class object.

b. Next, write an instance method called ***passOrFail()*** within the Student class. This method returns the string "Pass" if mark is greater than or equal to passingMark or "Fail" if mark is lower than passingMark.

c. Outside the class, instantiate a Student object called ***student1*** with ***name = 'John'*** and ***mark = 52*** as parameters, and use it to call the passOrFail() method. Assign the result to a variable called ***status1*** and print the value of status1.

d. Instantiate another Student object called ***student2*** with ***name = 'Jenny'*** and ***mark = 69*** as parameters, and use it to call the passOrFail() method. Assign the result to a variable called ***status2*** and print the value of status2.

Update the value of passingMark to 60 for all instances of the Student class and call the passOrFail() method for student1 and student2 again. Assign the results to status1 and status2 respectively and print the two values.

Question 3 (20 points)

Design a class **Message** that models an e-mail message. A message has a recipient, a sender, and a message text. Support the following methods:

- ✓ A **constructor** that takes the sender and recipient
- ✓ A method **get_sender** that returns the sender's name
- ✓ A method **get_recipient** that returns the recipient's name
- ✓ A method **append** that appends a line of text to the message body
- ✓ A method **toString** that makes the message into one long string like this: "From: Harry Morgan\nTo: Rudolf Reindeer\n . . ."

Write a program that uses this class to make a message and print it.

Question 4 (20 points)

Write a complete Python program that generates a list with 10 random integer numbers between 1 and 50. Then write a function called **substitute** that accepts the list as an argument and evaluates its elements. If an element is a multiple of 5, then that element will be substituted by its square. For example, if the first element of the list is 10, then it will be substituted by its square which is 100.

Sample Run:

Before substitution, the list is: [10 6 5 9 2 11 17 1 15 20]

After substitution, the list is: [100 6 25 9 2 11 17 1 225 400]

Question 5 (20 points)

Write a complete Python program that reads integers from the user until a blank line is entered. Once all of the integers have been entered by the user, your program should display all of the positive numbers, followed by all of the zeros, followed by all of the negative numbers, all in one line. Within each group, the numbers should be displayed in the same order that they were entered by the user. For example, if the user enters the values 3, -4, 1, 0, -1, 0, and -2 then your program should output the values 3, 1, 0, 0, -4, -1 and -2. Assume that the user does not enter invalid inputs.

Hint: Use lists to separate zeros, positive and negative numbers.

Sample Run

Enter an integer (blank to quit): 3
Enter an integer (blank to quit): -4
Enter an integer (blank to quit): 1
Enter an integer (blank to quit): 0
Enter an integer (blank to quit): -1
Enter an integer (blank to quit): 0
Enter an integer (blank to quit): -2
Enter an integer (blank to quit):

The numbers were:
3 1 0 0 -4 -1 -2