PGR211 - Advanced Programming for Data Science

Kristiania University College

Fall 2021

Examination paper released: November 10, 2021

Examination deadline: December 8, 2021

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 (2-3 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 the principles of object-oriented programming (e.g. class, object, method, inheritance, polymorphism, encapsulation, etc.) and can use these to write object-oriented code.

- understands the imperative, functional and object-oriented language features of Python and knows when it is appropriate to use each.

- demonstrates the understanding of web programming practices using Python language and can employ one of the popular web frameworks in Python (eg Flask, Django, etc.) for web programming.

- can test, debug, conduct version control (git) and code in teams.

Submit complete Python codes for Questions 1-4.

Question 1 (20 points)

a. Sensitive information is often removed, or redacted, from documents before they are released to the public. When the documents are released, it is common for the redacted text to be replaced with black bars. In this question, you will write a program that redacts all occurrences of sensitive words in a text file by replacing them with **asterisks**. Your program should redact sensitive words wherever they occur, even if they occur in the middle of another word. The list of sensitive words will be provided in a separate text file. Save the redacted version of the original text in a new file. The names of the original text file, sensitive words file, and redacted file will all be provided by the user.

b. Extend your program so that it redacts words in a case insensitive manner. For example, if **exam** appears in the list of sensitive words, then redact **exam**, **Exam**, **ExaM** and **EXAM**, among other possible capitalizations.

*** Note that your program should perform error checking.

*** You may find the ***replace*** method for string helpful for this question. Information about the ***replace*** method can be found on the Internet.

<u>Question 2 (20 points)</u>

A magic date is a date where the day multiplied by the month is equal to the two-digit year. For example, June 10, 1960 is a magic date because June is the sixth month, and 6 times 10 is 60, which is equal to the two-digit year.

a. Write a function named ***isMagicDate*** that determines whether or not a date is a magic date.
b. Use your function to create a main program that finds and displays all the magic dates in the 20th century.

In this question, you may need to have another function named ***daysInMonth*** that will take two parameters: **(1)** The month as an integer between 1 and 12, and **(2)** the year as a four digit integer and determines how many days there are in a particular month. Ensure that your function reports the correct number of days in "February" for **Leap Years**.

<u>Note</u>

Most years have 365 days. However, the time required for the Earth to orbit the Sun is actually slightly more than that. As a result, an extra day, February 29, is included in some years to correct for this difference. Such years are referred to as **<u>leap years</u>**. The rules for determining whether or not a year is a leap year follow:

- Any year that is divisible by 400 is a leap year.
- Of the remaining years, any year that is divisible by 100 is **not** a leap year.
- Of the remaining years, any year that is divisible by 4 is a leap year.
- All other years are **not** leap years.

Question 3 (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 4 (20 points)

Write a Python program using a function **def translate (dictionary, phone_number)** that takes a dictionary and a phone number (entered by the user) as arguments, translates the digits in phone number to words and **returns** it as a **string**. Consider the following two functions (main and translate) and write their definitions. Remember that the **translate** function should **return** a **string** and then the output will be printed in the **main function**.

Sample Run:

```
Enter your phone number: 93535352
Your phone number is: Nine Three Five Three Five Three Five Two
```

```
def main ():
```

```
def translate (dictionary, phone_number):
```

```
main ()
```

<u>Question 5 (10 points)</u>

Using your own words, compare Object Oriented Programming (OOP) and Functional Programming (FP) paradigms in Python. Discuss their differences and similarities in details.

<u>Question 6 (10 points)</u>

Using your own words, compare iterators, generators, and decorators in Python. You may need to provide examples to support your explanations.