Credit Risk: Exam 01

Complete Name:

Github Username:

Student ID:

Date:

Contents

Part 1: (60 pts) Theory

(6 pts) Git Concepts

Q1 (2 pts) What's the difference between using `git add` vs `git commit`?

Q2 Assume you are on a repository with one remote (origin) and with the following branches:

- `master`
- `working-branch-1`
- `working-branch-2`
- `solutions`

You are currently on `working-branch-2`. Do the following:

- **(0.5 pts)** Update your local `master` branch with the changes from the "origin" remote.
- **(0.5 pts)** Create a new branch from `master` named `working-branch-3`.
- **(0.5 pts)** Merge `solutions` into the new branch.
- **(0.5 pts)** Push your changes into the "origin" remote.

Q3 (2 pts) Assuming you are in a repository with multiple remotes:

- `origin`
- `upstream`

Each remote contains a branch named `common`. How can you update the `common` branch from `origin` with the latest changes from the `upstream` `common` branch?

(6 pts) Python Applications

Q4 (2 pts) What's a python virtualenv and why is it useful?

Q5 (1 pts) The python virtual machine uses a "stack-based" execution method. Describe how a "stack" works and briefly explain how python uses that data structure?

Hint: an stack-overflow error can occur over multiple recursive calls of the same function.

Q6 (1 pts for each) Name and describe at least 3 files (scripts) we use on a python project.

-
-
-

(15 pts) Object oriented programming

Q7 (1.5 pts for each) Name and explain the 4 object oriented programming principles:

-
-
-
-

Q8 (1 pts for each) Choose 2 principles and give a concrete example on how / when to use it:

-
-

Q9 (2 pts) Describe the difference between classes and objects:

Q10 (2 pts) Describe the difference between attributes and methods:

Q11 (2 pts) When should we use a static method over a regular method? Explain and give an example.

Q12 (1 pts) Where in the code can you make use of private methods?

(15 pts) OOP in Python

Consider the following `Human` class with two sub-classes named `Student` and `Professor`.

`Human` class definition:

```
import datetime as dt

STRING_FORMAT_DATE = "%Y-%m-%d"


class Human:

    def __init__(self, first_name, last_name, date_of_birth, **kwargs):
        self.date_of_birth = dt.datetime.strptime(date_of_birth, STRING_FORMAT_DATE)
        self.first_name = first_name
        self.last_name = last_name
        self.full_name = f"{first_name} {last_name}"
        self._kwargs = kwargs

    @property
    def age(self):
        today, dob = dt.datetime.today(), self.date_of_birth
        adjust = (today.month, today.day) < (self.dob.month, self.dob.day)
        return today.year - self.dob.year - adjust


    def greeting(self):
        raise NotImplementedError("Greeting method is not implemented")
```

Child class `Student` definition:

```
class Student(Human):
    notebook_name = "notes"

    @property
    def notes(self):
        return self._kwargs.get(self.notebook_name, "")

    def add_note(self, note):
        notes_content = self.notes + note
        self._kwargs[self.notebook_name] = notes_content

    def greeting(self):
        return "My name is {student_name} and I'm {student_age} years old.".format(
            student_name=self.full_name, student_age=self.age)
```

Child class `Professor` definition:

```
class Professor(Human):

    @property
    def lecture(self):
        return self._kwargs.get("lecture")

    def assign_lecture(self, lecture_name, override=False, fail=True):
        FAIL_MESSAGE = f"Cannot assign lecture {lecture_name} to professor " + \
                       f"{self.full_name} because {self.lecture} was previously assigned."
        if not self.lecture or override:
            self._kwargs["lecture"] = lecture_name
        elif not fail:
            print(FAIL_MESSAGE)
        else:
            raise ValueError(FAIL_MESSAGE)

    def greeting(self):
        return "I'm Prof. {professor_last_name} and {lecture_details}.".format(
            professor_last_name=self.last_name,
            lecture_details=f"I am teaching a lecture named '{self.lecture}'"
                if self.lecture else "I am currently not teaching any lecture")
```

Q13 (0.5 pts each) What are the common attributes between `Student` and `Professor`? Name at least 4.

-
-
-
-

Q14 (1 pts) Identify one common method between `Student` and `Professor`:

- Common method:

Q15 (1.5 pts for each) Identify one distinct method for each class:

- `Student`:
- `Professor`:

Q16 (3 pts) What does the `@property` decorator does in this context?

Given this code snippet:

```
# Create professor object
professor = Professor(
    first_name="Erwin",
    last_name="Schrödinger",
    date_of_birth="1887-08-12"
)

# A) First greeting
greeting_a = professor.greeting()

# B) Second greeting
professor.assign_lecture(lecture_name="Quantum Mechanics", fail=False)
greeting_b = professor.greeting()

# C) Third greeting
professor.assign_lecture(lecture_name="Probability Theory", fail=False)
greeting_c = professor.greeting()
```

Write out the value of the following variables:

- **Q17 (3 pts)** Value of `greeting_a`:
- **Q18 (3 pts)** Value if `greeting_c`:

(18 pts) FP in Python

Q19 (2 pts) In your own words, what's functional programming and why is it different than OOP?

Q20 (1 pts for each) What's the syntax to represent arbitrary positional arguments and named arguments on python functions?

- Positional arguments:
- Named arguments:

Q21 (1 pts) What are `lambda` functions in python?

Q22 (2 pts) What is a decorator and why are they useful?

Q23 (1 pts) What's the return type of a decorator?

Create a decorator that:

- **Q24 (2 pts)** Has the exact same arguments as any possible python function.
- **Q25 (1 pts)** Doesn't modify the behaviour of such function when applied (e.g. identity).

Given the following function:

```
import random

def get_random(a: int, b: int):
    return random.randint(a, b)
```

Q26 (3.5 pts) Create a decorator to re-try the function until it gets an odd number.

Q27 (3.5 pts) Create a meta-decorator that returns the nth power of the output number.

Part 2: (40 pts) Coding

(40 pts) flatten-json

Please follow the instructions on the `flatten-json` project.

- Project structure, best practices, and minimal functionality.
  - **Q28 (3 pts)** `(a)` - the `main.py` file is configured correctly (fire + logging)
  - **Q29 (6 pts)** `(b)` - the recursive `flatten_dict` function works as expected or at least on most cases.
  - **Q30 (6 pts)** `(c)` - the commands are reachable via the CLI and work as intended.
- **Q31 (25 pts)** Correct execution of all the examples AND the secret tests.