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 (1 pts) Assume 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?

Q4 (1 pts) In your own words, explain how git uses `hashes`.

(6 pts) Python Applications

Q5 (1 pts) In your own words, explain the python virtualenv and why is it useful?

Q6 (1 pts) In your own words, explain the python GIL and its limitations.

Q7 (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: a stack-overflow error can occur over multiple recursive calls of the same function.

Q8 (0.5 pts for each) Name and describe at least 3 files (scripts) we use on a python project (e.g., industry-crawler).

*

*

*

Q9 (0.5 pts for each) Using a creative example, explain the following concepts:

* `Concurrency`

* `Parallelism`

* `Distribution`

(15 pts) Object oriented programming

Q10 (1.5 pts for each) Name and explain the 4 object oriented programming principles:

*

*

*

*

Q11 (1 pts for each) Choose 2 principles and give a concrete example on how / when to use it:

*

*

Q12 (2 pts) Describe the difference between classes and objects:

Q13 (2 pts) Describe the difference between attributes and methods:

Q14 (2 pts) When should we use a static method over a regular method? Explain and give an example.

Q15 (1 pts) Briefly explain why is it recommended using the OOP on financial applications?

(15 pts) OOP in Python

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

`Human` class definition:

```python
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:

```python
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:

```python
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")
```

Q16 (0.5 pts each) What are the common attributes between `Student` and `Professor`? Name at least 4.

-
-
-
-

Q17 (1 pts) Identify one common method between `Student` and `Professor`:

- Common method:

Q18 (1.5 pts for each) Identify one distinct method for each class:

- `Student`:
- `Professor`:

Q19 (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:

* **Q20 (3 pts)** Value of `greeting_a`:

* **Q21 (3 pts)** Value if `greeting_c`:

(18 pts) FP in Python

Q22 (1.5 pts for each) Using your own understanding, name and explain the 4 functional programming principles:

-
-
-
-

Q23 (1 pts) Briefly explain the mathematical theory behind functional programming.

Q24 (1 pts) In your own words, explain the concept of a `monad`.

Q25 (1 pts) Explain at least one FP principle that the Apache Spark framework (distributed system) follows:

Q26 (1 pts) What's the difference between a narrow and a wide transformation on Apache Spark.

Q27 (1 pts) In your own words, explain why Apache Spark creates a DAG.

Q28 (0.5 pts for each) What's the syntax to represent arbitrary positional arguments and named arguments on python functions?

* Positional arguments:

* Named arguments:

Q29 (2 pts) What is a decorator and why are they useful?

Q30 (1 pts) What's the return type of a decorator?

Given the following function:

```
import random

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

Q31 (3 pts) Create a decorator to re-try the function until it gets an odd number.

Part 2: (40 pts) Coding

(40 pts) flatten-json

Please follow the instructions on the `flatten-json` python project located at `src/exams/flatten-json`. The results should be delivered via github (pull-request).

- Project structure, best practices, and minimal functionality.
  - **Q32 (3 pts)** `(a)` - the `__main__.py` and `main.py` files are configured correctly (fire + logging)
  - **Q33 (6 pts)** `(b)` - the recursive `flatten_dict` function works as expected or at least on most cases.
  - **Q34 (6 pts)** `(c)` - the commands are reachable via the CLI and work as intended.

- **Q35 (25 pts)** Correct execution of all the examples AND the secret tests.