Object Orientation Skills

Object Orientation Skills is meant to help you practice and assess your understanding of classes and object orientation in Python.

Follow the directions below to review what you should have learned, practice implementing classes, and submit an assessment of your understanding.

> **Note: Following Specifications**
>
> Throughout the assessment, remember to follow the specifications we provide carefully. Consistency in naming variables, methods, and attributes is an important part of software engineering.

Take Aways

- know how to define classes with attributes and methods
- understand the difference between instance attributes and class attributes
- know how to instantiate a class with initial parameters
- make a new class from an existing class via inheritance
- be able to explain abstraction, encapsulation, and polymorphism

Assessment

Complete Parts 1 through 5 in the file ***assessment.py***.

Be honest with yourself and your adviser about how this assessment goes for you. If you don't finish it, take note of which problems you weren't able to finish, and come back to them later. If you have *ideas* about how to solve something, but not a fully formed solution, leave your thoughts as comments in the function body.

We're assessing this on correctness and style — we want to see where you are on all of these things. So, choose good variable names, and show off good Python style. In addition, for each question, consider any "edge cases" where you might want to handle unusual input or errors and, as you are able to, think about how to handle those. If it makes sense to use a more intermediate concept, like a list comprehension, do so.

You must turn in all of your work **through FRODO** by **9PM on Sunday**.

Directions

Part 1: Discussion

Answer each question in your own words.

1. What are the three main design advantages that object orientation can provide? Explain each concept.
2. What is a class?
3. What is a method?
4. What is an instance in object orientation?
5. How is a class attribute different than an instance attribute? Give an example of when you might use each.

Part 2: Classes and Init Methods

Directions: Make Python classes that could store each of the following three pieces of data. Use the tables below as examples to guide you in creating class definitions for the following objects. Define an ***__init__*** method to allow callers of this class to provide initial values for each attribute. Note: these classes should all subclass ***object***; none is a subclass of any other.

1. Student

   Students can have first and last names and addresses. Here are two example students. Write a class that can store data such as this:

first_name	last_name	address
Jasmine	Debugger	0101 Computer Street
Jacqui	Console	888 Binary Ave

2. Question

   Questions include a question and a correct answer. Here are two example questions. Write a class that can store data such as this:

question	correct answer
What is the capital of Alberta?	Edmonton
Who is the author of Python?	Guido Van Rossum

3. Exam

   Notice that an Exam should have an attribute called ***questions***. Simply initialize the ***questions*** attribute as an empty list in the body ***__init__*** function. We'll deal with adding questions to the exam later on in this assignment. Your ***__init__*** function should take a ***name*** for the exam as a parameter.

   > **Note: A Note on Attributes**
   >
   > Though we've mainly seen attributes that are strings or integers, remember: attributes can also be lists and many other data types. In the case of our ***questions*** attribute, we'll have a list of ***Question*** objects.

For example, here are two exams. Make a class that could store data like this. Since the ***questions*** attribute will be a list of ***Question*** objects, let's call the questions above ***alberta_capital*** and ***python_author***. We might also imagine we have a couple Questions we instantiated called ***ubermelon_competitor*** and ***balloonicorn_color***. In that case, we could make exams with this data:

name	questions
Midterm	alberta_capital, python_author
Final	ubermelon_competitor, ballonicorn_color

Part 3: Methods

1. Add a method to the ***Exam*** class which takes a ***Question*** and adds it to the exam's list of questions.

   For example

```
$ python -i assessment.py
>>> set_q = Question(
...     'What is the method for adding an element to a set?')
...     '.add()')
>>> exam = Exam('midterm')
>>> exam.add_question(set_q)
```

2. Add a method to the ***Question*** class that prints the question to the console and prompts the user for an answer. It should return ***True*** or ***False*** depending on whether the correct answer matches the user's answer.

   For example

```
$ python -i assessment.py
>>> set_q.ask_and_evaluate()
What is the method for adding an element to a set? > .add()
True
```

3. Add a method to the ***Exam*** class which:

   - administers all of the exam's questions (how do you access each question in turn? Once you have a question, how do you administer it?), and
   - **returns** the user's tally of correct answers (as a decimal percentage)

   So, building on our code from problem 2, here's how the ***Exam*** class should work.

```
$ python -i assessment.py

>>> exam = Exam('midterm')
>>> set_q = Question(
...     'What is the method for adding an element to a set?',
...     '.add()')
```

```
>>> exam.add_question(set_q)
>>> pwd_q = Question(
...     'What does pwd stand for?',
...     'print working directory')
>>> exam.add_question(pwd_q)
>>> list_q = Question(
...     'Python lists are mutable, iterable, and what?',
...     'ordered')
>>> exam.add_question(list_q)
>>> exam.administer()
What is the method for adding an element to a set? > .add()
What does pwd stand for? > print working directory
Python lists are mutable, iterable, and what? > ummm...
66.66666666666666
```

Part 4: Create an Exam for a Student

> **Note: Reuse Your Classes (Yay!)**
>
> Part 4 doesn't require you to modify the class definitions you've created in the previous 3 parts of this assignment. You will, however, need to *use* the classes you've defined.

1. Write a *class* called ***StudentExam*** that has three methods: ***__init__*** and ***take_test***. This class should be able to store a student, an exam, and the student's score for that exam.

   The ***take_test*** method administers the exam and assigns the actual score to the ***StudentExam*** instance. This method should also print a message to the screen indicating the score; a return value isn't required.

2. Now, write a *function* (not a method) called ***example***, which does the following:

   - Creates an exam
   - Adds a few questions to the exam
     - These should be part of the function; no need to prompt the user for questions.
   - Creates a student
   - Instantiates a ***StudentExam***, passing the student and exam you just created as arguments
   - Administers the test for that student using the ***take_test*** method you just wrote

Part 5: Inheritance

A "quiz" is like an exam — it's a set of questions that students are prompted to answer. However, whereas exams are given a percentage score, quizzes are pass/fail. If you answered at least half of the questions correctly, you pass the quiz; otherwise, you fail. When you call the ***administer*** method on a quiz, it should return 1 if you passed or 0 if you failed.

Think about how you could solve this requirement: you have an ***Exam*** class and you want to have a ***Quiz*** class that is similar.

As you saw in Part 4 with ***StudentExam***, you will also need a new class that allows a student to take a quiz and stores the score received.

Write code to solve this problem. Incorporate as many of the “design” parts of the class lectures as you feel comfortable with.