Homework 0

CS 201 Data Structures 2

Spring 2023

The express purpose of this homework is to get you ready for your forthcoming assignments. It covers

- OOP in python
- Allowing native python functionality for your class

1. Introduction

You already know about OOP in C++. This homework will get you used to the corresponding syntax in python. Below is an example class in python.

```
1  class Car(object):
2      num_wheels = 4
3
4      def __init__(self, color):
5          self.wheels = Car.num_wheels
6          self.color = color
7
8      def drive(self):
9          if self.wheels <= Car.num_wheels:
10             return self.color + ' car cannot drive!'
11         return self.color + ' car goes vroom!'
12
13     def pop_tire(self):
14         if self.wheels > 0:
15             self.wheels -= 1
```

`Line 1` is a class declaration. By convention, all python classes that do not derive from any other class, must derive from the `object` class. Leaving out this inheritance does not make much difference. All subsequent lines that are indented become part of the definition of this class.

`Line 2` declares a *class variable*. That is, every object of this class has access to the same, single instance of this variable. This variable belongs to the class and not to any instance. Line 9 illustrates the corresponding syntax to access this variable. You may include class variables in your classes if needed.

Line 4 declares the *constructor*. The double underscores or *dunder*s in the function name are required. The constructor takes a mandatory parameter, `self`, which is a reserved word in python and automatically refers to the object that is calling this method. This is similar to `this` in C++. Just that `this` is passed implicitly in C++ whereas `self` *must* be the first parameter in the header of a method. Note that `self` does not have to be passed when the method is called. The constructor can take any number of additional parameters. In this case, it takes 1, `color`.

Lines 5 and 6 are the body of the constructor. You can put any desired logic here. Usually, the constructor initializes member variables, which is also the case here. Member variables need to be referred to using the `self` parameter and the *dot* syntax.

Lines 8 to 15 define other methods. Note the mandatory first parameter of `self` in each header, and how member variables are accessed. Line 9 additionally illustrates accessing a class variable. The methods here do not have any parameters other than `self`. This is a coincidence and not a requirement. That is, methods may have any number of parameters, just like regular functions. Just that the first parameter must be `self`. If the first parameter is not `self`, then the method belongs to the class, and not an instance.

Below is an example interaction with our class.

```
1  >>> my_car = Car('red')
2  >>> print(my_car.color)
3  red
4  >>> print(my_car.wheels)
5  4
6  >>> print(Car.num_wheels)
7  4
8  >>> print(my_car.num_wheels)
9  4
10 >>> print(Car.wheels)
11 Traceback (most recent call last):
12   File "<stdin>", line 1, in <module>
13 AttributeError: type object 'Car' has no attribute 'wheels'
14 >>> my_car.pop_tire()
15 >>> my_car.drive()
16 'red car cannot drive!'
17 >>> Car.drive()
18 Traceback (most recent call last):
19   File "<stdin>", line 1, in <module>
20 TypeError: drive() missing 1 required positional argument: 'self'
```

Line 1 creates an instance, `my_car`. This will internally call the `__init__` method of the `Car` class. Note that nothing needs to be passed for the `self` parameter of the `__init__` method.

Lines 2 to 5 illustrate access to member variables. Note that member variables are public in python!

Lines 5 to 13 illustrate access to class and member variables. The class variable, `num_wheels` , can be accessed via the class (line 6) and via an instance (line 8). However, a member variable cannot be accessed through the class (line 10).

Lines 14 to 20 illustrate calling member methods. They can be called via an instance using the dot syntax (lines 14 and 15) but not via the class (line 17). Note again that nothing needs to be explicitly passed to correspond to the `self` parameter of these methods.

Task: Solve the exercises here.

2. Disciples of Discipline

Having experienced this heavy semester before, we would like to transfer our wisdom and help you guys plan and execute your semester courses' study plan in a very structured and disciplined manner. Therefore, we want you to create a portal for managing all of your individual courses which would then help you design a study plan and prioritize study materials as per the commitments of a respective course. We also want you all to learn the art of self-sufficient learning where you are confident with internet scanning when it comes

to looking for answers for technical questions that one (and every born programmer) might encounter when coding. Thus, we will be providing you with links for concepts that you have to implement in this task; that you can consult, understand and then implement as per your requirements.

You will create a `Course` class to encapsulate all the important attributes for any course. It will also support functionality to add, remove, and print assignments. You will also create a `Schedule` class which will contain your courses and indicate your schedule and total number of credits this semester.

Task: Implement the methods in the `Course` class provided in the accompanying file, `src/course.py`.

Task: Implement the methods in the `Schedule` class provided in the accompanying file, `src/schedule.py`.

Resources: The classes make use of the `datetime.datetime` object which you can learn about here.

3. Guten Appetit

You are a celebrated chef with many famous dishes to your name. A believer in open source, you are now looking to share your inventions with the world. A computer scientist at heart, you are developing a library which you will use to program your recipes. You will eventually disseminate this library so that not only do people learn of your recipes but they can use the library to program and share their recipes as well.

Two of the classes in your library are `Ingredient` and `Recipe`. `Ingredient` stores a name. `Recipe` stores a name and `Ingredient` objects along with the required quantity in grams. Below is an example interaction with these classes.

```
>>> my_recipe = Recipe('Stairway to Heaven')
>>> my_recipe.add_ingredient(Ingredient('Sugar'), 10)
>>> my_recipe.add_ingredient(Ingredient('Spice'), 10)
>>> my_recipe.add_ingredient(Ingredient('Everything Nice'), 20)
>>> my_recipe.add_ingredient(Ingredient('Chemical X'), 100)
>>> len(my_recipe)  # number of ingredients
4
>>> for ing, amt in my_recipe:  # iterate over ingredients
...     print(f'{ing}: {amt}g')
...
Sugar: 10g
Spice: 10g
Everything Nice: 20g
Chemical X: 100g
```

Task: Write both classes in the accompanying file, `src/recipe.py`, such that they support this interaction.

Resources: There are several features you will need to learn about.

- Writing a custom iterator: See here (first 3 sections only) for an insightful explanation and here for a quick example.
- Calling `len()` on a custom class: Included in the above references..
- Tuple assignment and unpacking: See here.

viel Spaß