

**College of Engineering**

## CS 1337/1337L Introduction to Object-Oriented Programming

## Dr. Fatemeh (Azi) Tabei

## Fall 2021

**Lab - 10/19/2021**

**Due:10/19/2021**

Please upload your assignments through your WTClass on time. The assignment will NOT be accepted if the time is passed or not submitted properly through the WTClass. It is your responsibility to make your submissions before the deadline.

**Introduction**

Here's an example of a Car class:

class Car(object):

num\_wheels = 4

def \_\_init\_\_(self, color):

self.wheels = Car.num\_wheels

self.color = color

def drive(self):

if self.wheels <= Car.num\_wheels:

return self.color + ' car cannot drive!'

return self.color + ' car goes vroom!'

def pop\_tire(self):

if self.wheels > 0:

self.wheels -= 1

* **class**: a blueprint for how to build a certain type of object. The Car class (shown above) describes the behavior and data that all Car objects have.
* **object**: a particular occurrence of a class. In Python, we create object (instances) of a class like this:

my\_car = Car('red')

* **attribute** or **field**: a variable that belongs to the class. Think of an attribute as a quality of the object: cars have *wheels* and *color*, so we have given our Car class self.wheels and self.color attributes. We can access attributes using **dot notation**:

>>> my\_car.color

'red'

>>> my\_car.wheels

4

* **method:** Methods are just like normal functions, except that they are tied to an instance or a class. Think of a method as a "verb" of the class: cars can drive and also pop their tires, so we have given our Car class the methods drive and pop\_tire. We call methods using dot notation:

>>> my\_car = Car('red')

>>> my\_car.drive()

'red car goes vroom!'

* **constructor**: As with data abstraction, constructors describe how to build an instance of the class. Most classes have a constructor. In Python, the constructor of the class defined as \_\_init\_\_. For example, here is the Car class's constructor:

def \_\_init\_\_(self, color):

self.wheels = Car.num\_wheels

self.color = color

The constructor takes in one argument, color. As you can see, the constructor also creates the self.wheels and self.color attributes.

self: in Python, self is the first parameter for many methods (in this class, we will only use methods whose first parameter is self). When a method is called, self is bound to an instance of the class. For example:

>>> my\_car = Car('red')

>>> car.drive()

Notice that the drive method takes in self as an argument, but it looks like we didn't pass one in! This is because the dot notation *implicitly* passes in car as self for us.

**Exercise Create an object-oriented Movie List program**

In this exercise, you’ll convert the Movie List program presented in chapter 6 so it uses objects instead of storing the data for the movie in a list.

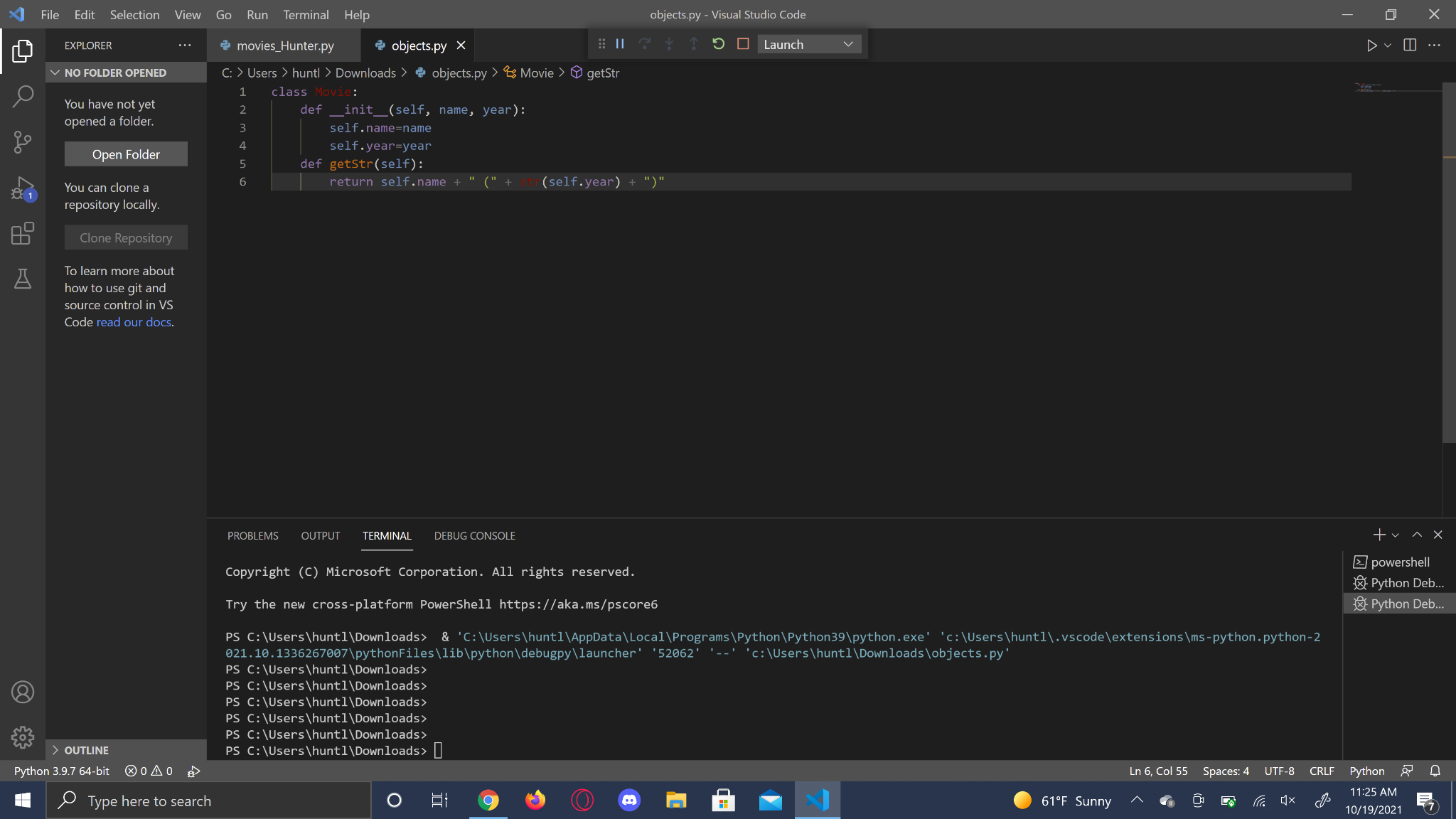
**Open and test the program**

1. In IDLE, open the movie.py file that’s in the assignment folder.
2. Review the code and note how it uses a list to store the data for each movie.
3. Run the code to make sure it works correctly.

**Define a Movie object that can store the data for each movie**

4. Add a module named objects to the program’s folder on your computer.

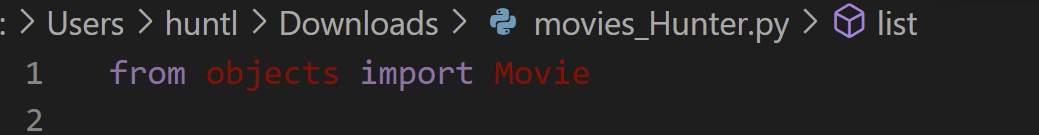
5. In the object module, write the code for a class named Movie that defines a Movie object that stores the name and year of a movie. This class should include a getStr() method that returns the name of the movie followed by its year in parentheses.



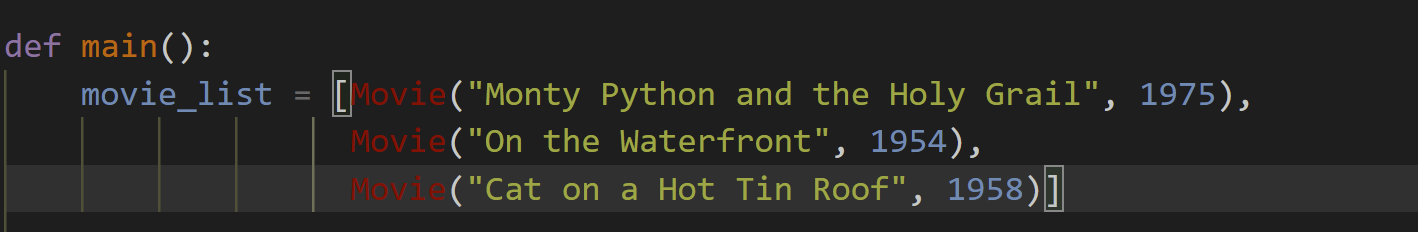
6. Run the module. This should display the interactive shell. Then, use the shell to test the class by creating a Movie object and printing it to the console.

**Modify the program so it uses the Movie object**

7. Switch back to the movie.py file, and add a statement that imports the Movie class. Save your new movie file as **movie\_StudentName.py**.



8. Modify the main() function so it creates a list of Movie objects instead of a list of lists.



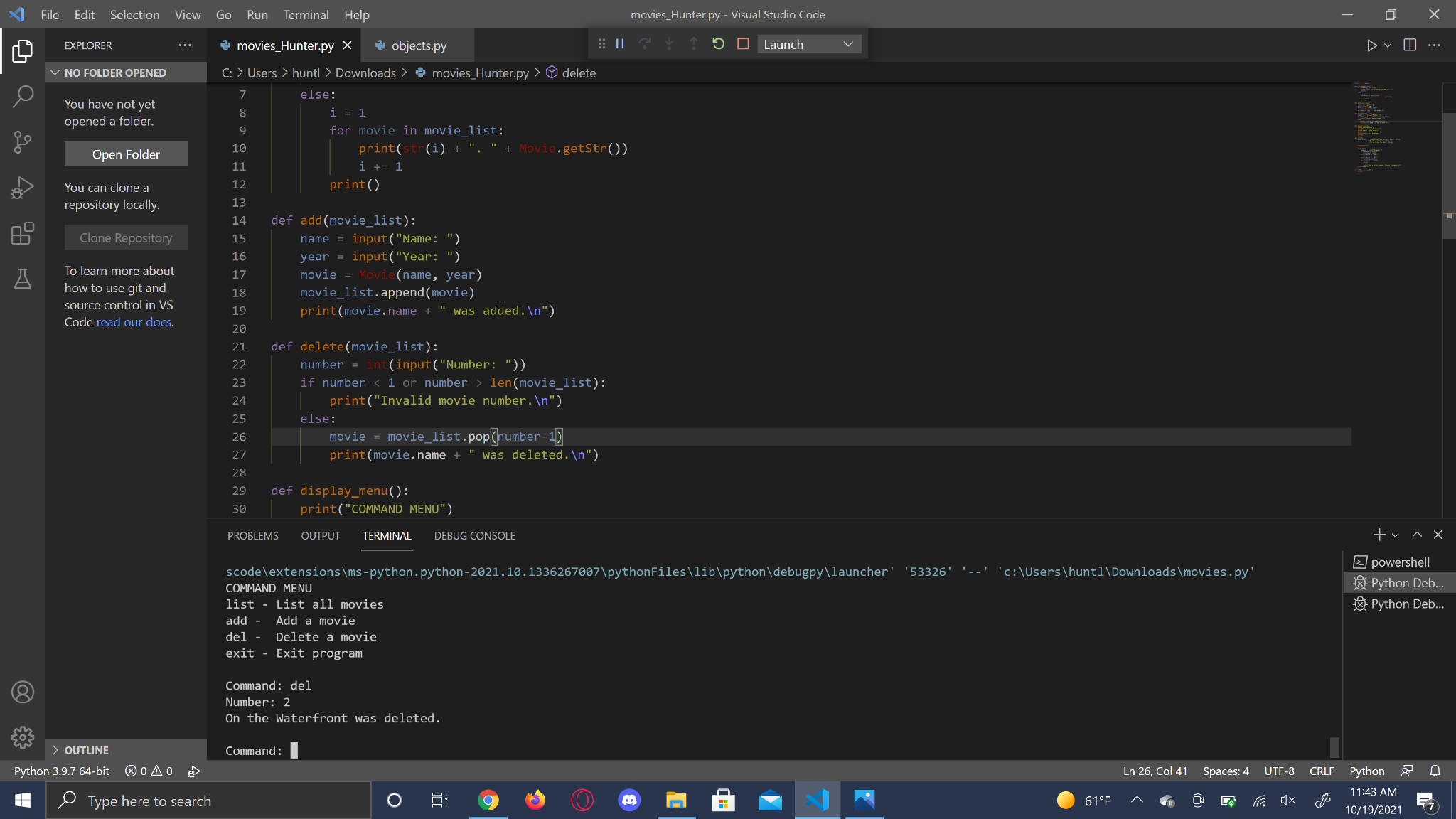
9. Modify the list() function so it uses the getStr() method to display each movie. Note how this simplifies the code.



10. Modify the add() function so it creates a Movie object from the user input. Note how this simplifies the code.

On 11

11. Modify the delete() function so it uses the Movie object to display the name of the movie that’s deleted. Note how this makes the code easier to read and understand.



\*\*Please upload both the file of the object and modified movie program movie\_StudentName.py. Please make sure to test your code before uploading.