You will need to scan and upload the pdf for just the first problem this week. The following two problems have corresponding templates ready for you to get started on Github. Do not forget to adjust the README to indicate you have completed the assignment before your final commit!

Get Assignment link: https://classroom.github.com/a/eOR0YH6W

1. Below are some snippets of code along with what their author was hoping to achieve. For each determine if the output would be as desired. If something is wrong, write out specifically how you could fix the problem while still meeting their desired objective.

   (a) **Objective:**

   To construct a class to hold an individual address. The `print_addr` method should print out the address in a nice fashion.

   Code:

```python
class Address:
    def __init__(self, number, street, city, state, _zip):
        self.number = number
        self.street = street
        self.city = city
        self.state = state
        self.zip = _zip

    def print_addr():
        print(f'{self.number} {self.street}')
        print(f'{self.city}, {self.state}')
        print(f'{self.zip}')
```

(b) **Objective:**

To construct a class to hold information on a movie. The method `is_better` should compare the IMDB scores of two movies and return the title of whichever is higher.

Code:

```
class Movie:
    def __init__(self, title, year, imdb_score):
        self.title = title
        self.year = year
        self.score = imdb_score

    def is_better(self, other_movie):
        if self.score > other_movie.score:
            return self.title
        else:
            return other_movie.title
```

(c) **Objective:**

To construct a class to hold information on a car. When the car object is printed, the user wants the various data attributes about the car to be summarized in a nice format.

Code:

```
class Car:
    def __init__(self, make, model, color, year)
        self.make = make
        self.model = model
        self.color = color
        self.year = year

    def __str__(self):
        p1 = f'A {self.color} {self.make} {self.model}'
        p2 = f'made in the year {self.year}.'
        print(p1,p2)
```

2. Here you will be constructing a Queue class for Python. Queues are a fundamental computer science data structure, and operate just like a line at an amusement park. You add elements to a queue and they maintain their order. When you remove elements from a queue, you remove the element that has been in the queue the longest (aka, "first-in-first-out", or FIFO). In other words, you add new people to the back of the line and you remove people from the front of the line.

   In the file `Prob2.py`, the very basic skeleton of a Queue class has been started for you. You will need to add three methods to get a bare-bones functioning Queue:

   - `__init__`: to initialize your Queue. How would you like to store the queue's elements? What would be a good choice? You won't need to pass any extra parameters in here, just default the Queue to starting out empty. Call your data attribute for the Queue `self.q`.
   - `add`: adds one element to the Queue
   - `remove`: removes one element from the Queue and returns it. If the queue is empty, a message saying: "The queue is empty!" should be returned.

   You can test your class once it is complete. Some results might look like:

```
>> q = Queue()
>> q.add(2)
>> q.add(4)
>> q.remove()
2
>> q.add(6)
>> q.remove()
4
>> q.remove()
6
>> q.remove()
'The queue is empty!'
```

3. In this problem you'll be constructing a class to handle fractions. We want to try to build as much functionality into the class as possible, so everything from controlling how class objects get printed to properly doing math with fractions.

   (a) The class `Fraction` is started for you in `Prob3.py`. Add an `__init__` method which takes a numerator and a denominator and stores them as data attributes. In addition, it will help a lot in troubleshooting this problem if you have a clean way to print the fraction to the screen. So add another method which will ensure that when you print a Fraction object to the screen, it displays nicely. An example of creating and instance of Fraction and printing it might look like:

```
>> A = Fraction(2,3)
>> print(A)
2/3
```

   (b) One thing that is frequently necessary when dealing with fractions is reducing them into their simplest possible form. So simplifying $\frac{2}{6}$ to be $\frac{1}{3}$ for example. Add a method of your class called `reduce`, which will take a `Fraction` object and return a corresponding *new* `Fraction` object in simplest form. To do this, it may be helpful to calculate the greatest common divisor between two values. We've looked at ways to do this already earlier this semester, so feel free to consult what we did then or look up and implement a different algorithm to achieve the desired result.

   (c) Now lets add a few very simple methods just to give our class some more flexibility. In general, you can write any fraction as a decimal number. If then a user calls `float()` on an object that is an instance of `Fraction`, it would make sense that we return a float value. To override the functionality of `float`, the special method name is `__float__` (fittingly).

   Taking the inverse of a fraction is also frequently useful, so go ahead and add a method called `inverse` which will return the inverse of a `Fraction` object.

   (d) Finally, we should add functionality for at least basic multiplication and division. The corresponding special method names are given in the table below for the different operations.

`A*B`	`A.__mul__(B)`	Multiplication
`B*A`	`A.__rmul__(B)`	Multiplication
`A/B`	`A.__truediv__(B)`	Normal division

   We won't worry about instances where you are trying to multiply or divide a fraction by a float here. But multiplying or dividing a fraction by an integer should be handled gracefully by your program!