Purpose

The goal of this lab is to introduce you to classes in Python, and to give you some practice writing them.

Preparation

Remind yourself about complex numbers. Read chapters 15 and 16 in the textbook.

Assignment

1. Write a class `Circle`, whose constructor takes a single argument, representing the radius. Put the class definition in a file called `circle.py`

2. Write a member function, called `area`, for the class that returns the area of the circle.

3. Write a member function, called `diameter`, that returns the circle diameter. Write a member function, called `circumference` that returns the circumference of the circle.

4. Now we're going to write a class for complex numbers. Start by writing a class called `Complex` that takes two arguments, representing the real and imaginary parts of a complex number. Put the class definition in a file called `complex.py`, and call the member variables `re` and `im`. You should make these arguments optional, and default to something sensible if they are missing. Check that your code does the right thing in response to the following:

   ```
   a = Complex(1.0, 2.3)    # 1 + 2.3i
   b = Complex(2)           # 1 + 0i
   c = Complex()            # 0 + 0i
   ```

5. Implement the `__str__()` function, so that

   ```
   a = Complex(1, 2)
   b = Complex(1, -2)
   print a
   print b
   ```

   works and prints out

   ```
   (1 + 2i)
   (1 - 2i)
   ```

6. Implement addition, such that the following works:

```
a = Complex(1, 2)
b = Complex(3, 4)
print a + b
print a + 1
print 1 + a
```

and prints out

```
(4 + 6i)
(2 + 2i)
(2 + 2i)
```

Once you have addition working, implement subtraction. This should look a lot like addition.

7. Now, implement multiplication and division. Verify that everything works as expected.

8. Finally, implement the complex conjugate:

```
a = Complex(1, 2)
print ~a
```

should print out

```
(1 - 2i)
```

9. OK. Let's use your new class. Write a new function, in a file called roots.py, that calculates the roots of a quadratic function. You should call it like this:

```
roots(1, 2, 3)     # This is x^2 + 2x + 3
```

and it should return a list of all of the roots of the function. Sometimes these will be real numbers, sometimes the will be complex. Sometimes there will be two roots, and sometimes there will be one, or zero, depending on the values you pass in. You should deal with all of these cases. You should return a tuple containing all the roots you find.

Grading

If you use the built-in complex number implementation in your code, you will get zero points for this lab. The goal of this lab is for you to write your own code for complex numbers. You can, however, use the built-in implementation to test your own code.

There are 25 points available in this lab.

1. 2 points for a correctly-defined circle class. 1 point each for the area, diameter, and circumference functions. [5 points total]

2. 2 points for a correctly-defined complex number class. 1 point for handling fewer than 2 arguments in the constructor. [3 points total]

3. 1 point for correctly implementing `__str__`. 1 point for dealing correctly with a negative imaginary term. [2 points total]

4. 1 point for being able to add two complex numbers correctly. 1 point for being to add a complex to a number. 1 point for being able to add a number to a complex. 1 point for being able to do all three for subtraction. [4 points total]

5. 1 point for being able to multiply two complex numbers correctly. 1 point for being to multiply a complex by a number. 1 point for being able to multiply a number by a complex. 1 point for being able to do all three for division. [4 points total]

6. 1 point for the complex conjugate. [1 point]

7. 1 point for a well-defined roots function. 1 point for handling two real roots. 2 points for handling 1 complex roots. 1 point for handling the 1 root case. 1 point for handling the zero root case. [6 points total]

Thoughts

1. Support for complex numbers already exists in Python (of course). However, for the purposes of this lab, were going to assume that it doesn't and we're going to write out own code. You could, however, use the built-in implementation to test the correctness of your own implementation, if you like.

2. There seems like a lot of code to write this week. It's actually less that you think since once you figure out how the operators work, most of the functions look more-or-less the same.

3. We're going to ask you do implement a bunch of mathematical functions without telling you how. All of these can be implemented as special class member functions, which are listed here (Links to an external site.). Only implement the ones you need.

4. Addition and multiplication of complex numbers both commute. This means that you can define one version of the function in terms of another. For example, the function `__radd__` should call the function `__add__`.

5. Some of the functions, like addition, will have different types in the second argument for different calls. The right way to deal with this is through duck typing. Assume it's a Complex number, and do the calculation. Wrap this in a `try`-`except` block, and deal with the non-Complex case (where it's a number) in the `except` block.

The Rules

Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.