Complex Number

Problem ID: a19p02complexnumber

The main file, which handles input and output, is already provided. - Please only submit your class definition, without any code outside the class!

In mathematics, a complex number is an element of a number system that extends the real numbers with a specific element denoted i, called the imaginary unit and satisfying the equation $i \cdot i = -1$. Every complex number can be expressed in the form $a + bi$, where a and b are real numbers. Example: $5 - 2i$, with $a = 5$ and $b = -2$. For the complex number $a + bi$, we say a is the real part while b is called the imaginary part.

Write a class called `ComplexNumber`, which has two attributes, real part, and imaginary part, initialized with parameters which both have a default value of 0. We will leave it to you to decide how you want to store these values internally.

The class should implement the functionality detailed below. We recommend that you implement it in the order listed here, running your code against the unit tests after each step and making sure to pass the corresponding tests. There is some starter code in the template to get you started.

1. Conversion to a string should work as follows:

```
>>> z = ComplexNumber(5, 7)
>>> print(z)
5 + 7i
>>> str(z)
'5 + 7i'
>>> str(ComplexNumber(5, -7))
'5 - 7i'
>>> str(ComplexNumber(5, 0))
'5'
>>> str(ComplexNumber(5))
'5'
>>> str(ComplexNumber(-5))
'-5'
>>> str(ComplexNumber(0, 7))
'7i'
>>> str(ComplexNumber(0, -7))
'-7i'
>>> str(ComplexNumber(0, 0))
'0'
>>> str(ComplexNumber())
'0'
```

2. The class should implement the `__repr__()` method in a similar way as we did with the Vector class in Assignment 18:

```
>>> z = ComplexNumber(5, 7)
>>> repr(z)
'ComplexNumber(5, 7)'
>>> w = eval(repr(z))
>>> type(w) == ComplexNumber
True
>>> print(w)
5 + 7i
```

3. The class should implement methods called `re()` and `im()` for getting the real part and the imaginary part:

```
>>> z = ComplexNumber(3, 4)
>>> z.re()
3
```

```
>>> z.im()
4
```

4. Two complex numbers should be considered equal if, and only if, their real parts are equal and their imaginary parts are equal. Here we will not bother with floating point errors like in Assignment 18.

```
>>> z = ComplexNumber(3, 4)
>>> w = ComplexNumber(5, 4)
>>> z == w
False
>>> z == z
True
>>> z == ComplexNumber(3, 4)
True
>>> z == eval(repr(z))
True
```

5. The class should implement a method called `conjugate()`. The conjugate of a complex number $a + bi$ is defined as $a - bi$.

```
>>> z = ComplexNumber(3, 4)
>>> w = z.conjugate()
>>> type(w) == ComplexNumber
True
>>> print(w)
3 - 4i
>>> print(w.conjugate())
3 + 4i
>>> z.conjugate().conjugate() == z
True
```

6. The class should also implement the `__neg__()` method:

```
>>> z = ComplexNumber(5, -3)
>>> type(-z) == ComplexNumber
True
>>> print(-z)
-5 + 3i
>>> print(-(-z))
5 - 3i
>>> z == -(-z)
True
```

7. Addition, subtraction and multiplication of complex numbers can be naturally defined by using the rule $i \cdot i = -1$ combined with the associative, commutative and distributive laws:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$
$$(a + bi) - (c + di) = (a - c) + (b - d)i$$
$$(a + bi) \cdot (c + di) = (a \cdot c - b \cdot d) + (a \cdot d + b \cdot c)i$$

The class should overload the $+$, $-$ and $\star$ operators:

```
>>> z = ComplexNumber(300, 800)
>>> w = ComplexNumber(1, -2)
>>> type(z + w) == ComplexNumber
True
>>> print(z + w)
301 + 798i
```

```
>>> type(z - w) == ComplexNumber
True
>>> print(z - w)
299 + 802i
>>> type(z * w) == ComplexNumber
True
>>> print(z * w)
1900 + 200i
```

8. Every nonzero complex number also has a multiplicative inverse:

$$\frac{1}{a+bi} = \frac{a}{a^2+b^2} - \frac{b}{a^2+b^2}i$$

and thus division of complex numbers can be defined as well:

$$\frac{z}{w} = z \cdot \frac{1}{w}$$

```
>>> z = ComplexNumber(4, 3)
>>> type(z.inverse()) == ComplexNumber
True
>>> print(z.inverse())
0.16 - 0.12i
>>> w = ComplexNumber(1) / z
>>> type(w) == ComplexNumber
True
>>> print(w)
0.16 - 0.12i
>>> print(z.inverse().inverse())
4.0 + 3.0i
>>> z == z.inverse().inverse()
True
```

The method for overloading the `/` operator is `__truediv__()` (in python 3).

There is much more that can be done with complex numbers, such as mixing them with other numeric types in arithmetic operations, (`int`, `float`, etc.), calculating their polar coordinates, taking powers and roots of complex numbers, etc. But we will leave that as a further exercise for those of you who are interested.

Testing

Provided is a test file with unit tests that you can and should use to test your code locally before submitting your solution. It will provide you with quicker feedback than Gradescope does, and more detail on what is going wrong, if anything.

These are the old test cases from before we moved the assignments to Gradescope. The autograder on Gradescope uses different tests that are randomly generated and more numerous, and does not use exatly the same types of tests as the ones you can see in the old unit test file.

Below we include a description of the format of the test cases in the autograder, how the input and output is structured.

This is just FYI, and mostly serves as a documentation for us, the authors. You do not really need to worry about these details, because you do not need to write any code for dealing with this input, that is something that the autograder handles for you. We only include this description to explain what it is you are seeing in the samples, because we can not include test cases that are not secret (samples) without them appearing in the pdf automatically.

Input

In the input files for the test cases, the first line states the type of test case, which method the test is aimed at testing.

Then, depending on whether the method being tested deals with one or two complex numbers, there follows either one or two block of lines specifying sets of parameters for the constructor. Each block begins with one line stating

how many arguments are specified, either zero, one or two. Then follow that many lines with values for the specified parameters, one value per line, indented a little bit for easier human readability. So each block will contain 1 to 3 lines.

Those parameters for which no value is given will rely on the defaults.

Output

The ouput should be rather self-explanatory.

Note that in test cases dealing with division or inversion, the autograder will insert a space in front of the i. This is not something that your class needs to do, we just do this so that the tokenizer can see the coefficient as a separate token, and deal with it as a number rather than just any other string. Then the autograder can use floating point tolerance to compare the number from your solution to the one from our solution.

We only need to do this in the case of inversion and division, because all test cases are restricted to integers, so the only floating point errors that can come up are in cases where division is involved.

Sample Input 1

```
__str__
0
```

Sample Output 1

```
Initializing ComplexNumber with 0 specified parts,
and using default values for the remaining 2 parts.
Complex number created without errors.
0
```

Sample Input 2

```
__str__
1
 5
```

Sample Output 2

```
Initializing ComplexNumber with 1 specified parts,
and using default values for the remaining 1 parts.
 re = 5
Complex number created without errors.
5
```

Sample Input 3

```
__str__
2
 5
 7
```

Sample Output 3

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 5
 im = 7
Complex number created without errors.
5 + 7i
```

Sample Input 4

```
__str__
2
 5
 -7
```

Sample Output 4

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 5
 im = -7
Complex number created without errors.
5 - 7i
```

Sample Input 5

```
__str__
2
 -5
 -7
```

Sample Output 5

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = -5
 im = -7
Complex number created without errors.
-5 - 7i
```

Sample Input 6

```
conjugate
2
 5
 7
```

Sample Output 6

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 5
 im = 7
Complex number created without errors.
The conjucate of 5 + 7i is:
5 - 7i
```

Sample Input 7

```
inverse
2
 3
 4
```

Sample Output 7

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 3
 im = 4
Complex number created without errors.
The inverse of 3 + 4i is:
0.12 - 0.16 i
```

Sample Input 8

```
__add__
2
 3
 4
2
 5
 7
```

Sample Output 8

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 3
 im = 4
Complex number created without errors.
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 5
 im = 7
Complex number created without errors.
Result of adding 3 + 4i and 5 + 7i together:
8 + 11i
```

Sample Input 9

```
__mul__
2
 3
 4
2
 5
 7
```

Sample Output 9

```
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 3
 im = 4
Complex number created without errors.
Initializing ComplexNumber with 2 specified parts,
and using default values for the remaining 0 parts.
 re = 5
 im = 7
Complex number created without errors.
Result of multiplying 3 + 4i and 5 + 7i:
-13 + 41i
```

Sample Input 10	Sample Output 10
__truediv__ 2 5 7 2 3 4	Initializing ComplexNumber with 2 specified parts, and using default values for the remaining 0 parts. re = 5 im = 7 Complex number created without errors. Initializing ComplexNumber with 2 specified parts, and using default values for the remaining 0 parts. re = 3 im = 4 Complex number created without errors. Result of dividing 5 + 7i by 3 + 4i: 1.7200000000000002 + 0.039999999999999925 i