Formatting Complex Numbers

For a project in one of my classes we have to output numbers up to five decimal places.It is possible that the output will be a complex number and I am unable to figure out how to output a complex number with five decimal places. For floats I know it is just:

```
print "%0.5f"%variable_name
```

Is there something similar for complex numbers?

python formatting number-formatting complex-numbers

edited Jan 1 at 15:35
Cœur
5,942 1 42 61

asked Oct 12 '11 at 20:27
island_hopper
100 1 8

5 Answers

For questions like this, the Python documentation should be your first stop. Specifically, have a look at the section on string formatting. It lists all the string format codes; there isn't one for complex numbers.

What you can do is format the real and imaginary parts of the number separately, using `x.real` and `x.imag` , and print it out in `a + bi` form.

answered Oct 12 '11 at 20:34
David Z
74.5k 11 170 200

2 Just to note, in Python the imaginary part uses `j` . – Peter Wood Jun 4 '14 at 9:29

```
>>> n = 3.4+2.3j
>>> n
(3.4+2.3j)
>>> '({0.real:.2f} + {0.imag:.2f}i)'.format(n)
'(3.40 + 2.30i)'
>>> '({c.real:.2f} + {c.imag:.2f}i)'.format(c=n)
'(3.40 + 2.30i)'
```

To handle both positive and negative imaginary portions properly you would need a even more complicated formatting operation:

```
>>> n = 3.4-2.3j
>>> n
(3.4-2.3j)
>>> '({0:.2f} {1} {2:.2f}i)'.format(n.real, '+-'[n.imag < 0], abs(n.imag))
'(3.40 - 2.30i)'
```

Update

Although you *can't* use `f` as a presentation type for complex numbers using the string formatting operator `%` :

```
n1 = 3.4+2.3j
n2 = 3.4-2.3j

try:
    print('test: %.2f' % n1)
except Exception as exc:
    print('{}: {}'.format(type(exc).__name__, exc))
```

Output:

```
    TypeError: float argument required, not complex
```

You *can* use it with complex numbers via the `str.format()` method. This isn't explicitly documented, but is implied by the String Formatting Operations documentation which just says:

> `'f'` Fixed point. Displays the number as a fixed-point number. The default precision is `6` .

. . .so it's easy to overlook. In concrete terms, the following works in both Python 2.7.10 and 3.4.3:

```
print('n1: {:.2f}'.format(n1))
print('n2: {:.2f}'.format(n2))
```

Output:

```
n1: 3.10+4.20j
n2: 3.10-4.20j
```

This doesn't give you quite the control the code in my original answer does, but it's certainly much more concise (and handles both positive and negative imaginary parts).

edited Jun 18 '15 at 20:28 answered Jan 27 '13 at 14:05
martineau
43.8k 6 62 96

What if the imaginary part is negative? – Peter Wood Jun 4 '14 at 9:32

@Peter: To handle both positive and negative imaginary portions you could use `'({0.real:.2f}{0.imag:+.2f}j)'.format(n)` . – martineau Jun 4 '14 at 10:23

That would produce, for example, `(3.40 -2.30j)` . What if you want `(3.40 - 2.30j)` ? – Peter Wood Jun 4 '14 at 10:46

@Peter: For that you'd need something like `'({0:.2f} {1} {2:.2f}j)'.format(n.real, '+' if n.imag >= 0 else '-', abs(n.imag))` . If you have further questions please post them as such rather than as comments. – martineau Jun 4 '14 at 12:03

Sorry, I was trying to help you correct and improve your answer. With a negative imaginary part it will print, for example, `(3.49 + -2.30j)` . – Peter Wood Jun 4 '14 at 12:11

```
>>> n = 3.4 + 2.3j
>>> print '%05f %05fi' % (n.real, n.imag)
3.400000 2.300000i
```

answered Oct 12 '11 at 20:37
Facundo Casco
4,959 3 30 52

Changed to `'%05f %+05fi' % (n.real, n.imag)` it always shows the imaginary part's sign which looks mathematically nice in both cases: `3.400000 +2.300000i` and `3.400000 -2.300000i` . – flonk Feb 5 '14 at 15:02

Neither String Formatting Operations - i.e. the modulo (`%`) operator) - nor the newer `str.format()` Format String Syntax support complex types. However it is possible to call the `__format__` method of all built in numeric types directly. Here is an example:

```
>>> i = -3 # int
>>> l = -33L # long (only Python 2.X)
>>> f = -10./3 # float
>>> c = - 1./9 - 2.j/9 # complex
>>> [ x.__format__('.3f') for x in (i, l, f, c)]
['-3.000', '-33.000', '-3.333', '-0.111-0.222j']
```

Note, that this works well with negative imaginary parts too.

edited Sep 21 '14 at 4:58 answered Sep 7 '14 at 7:17
martink

21 2

As of Python 2.6 you can define how objects of your own classes respond to format strings. So, you can define a subclass of `complex` that can be formatted. Here's an example:

```
>>> class Complex_formatted(complex):
...     def __format__(self, fmt):
...         cfmt = "({:" + fmt + "}{:+" + fmt + "}j)"
...         return cfmt.format(self.real, self.imag)
...
>>> z1 = Complex_formatted(.123456789 + 123.456789j)
>>> z2 = Complex_formatted(.123456789 - 123.456789j)
>>> "My complex numbers are {:0.5f} and {:0.5f}.".format(z1, z2)
'My complex numbers are (0.12346+123.45679j) and (0.12346-123.45679j).'
>>> "My complex numbers are {:0.6f} and {:0.6f}.".format(z1, z2)
'My complex numbers are (0.123457+123.456789j) and (0.123457-123.456789j).'
```

Objects of this class behave exactly like `complex` numbers except they take more space and operate more slowly; reader beware.

edited Aug 2 '14 at 23:36

answered Aug 2 '14 at 23:27

FutureNerd
509 4 9