Python object.__repr__(self) should be an expression?

I was looking at the builtin object methods in the Python documentation, and I was interested in the documentation for `object.__repr__(self)` . Here's what it says:

> Called by the repr() built-in function and by string conversions (reverse quotes) to compute the "official" string representation of an object. If at all possible, this should look like a valid Python expression that could be used to recreate an object with the same value (given an appropriate environment). If this is not possible, a string of the form <...some useful description...> should be returned. The return value must be a string object. If a class defines **repr**() but not **str**(), then **repr**() is also used when an "informal" string representation of instances of that class is required.
>
> This is typically used for debugging, so it is important that the representation is information-rich and unambiguous

The most interesting part to me, was...

> If at all possible, this should look like a valid Python expression that could be used to recreate an object with the same value

... but I'm not sure exactly what this means. It says it should *look* like an expression which can be used to recreate the object, but does that mean it should just be an example of the sort of expression you could use, or should it be an actual expression, that can be executed (eval etc..) to recreate the object? Or... should it be just a rehasing of the actual expression which was used, for pure information purposes?

In general I'm a bit confused as to exactly what I should be putting here.

python

asked Jan 16 '09 at 22:37


Fara
3,281 1 11 25

This question is linked to from the official course notes on MIT's online 6.00.1x: Introduction to Computer Science. – dotancohen Jul 27 at 19:46

add a comment

4 Answers

```
>>> from datetime import date
>>>
>>> repr(date.today())        # calls date.today().__repr__()
'datetime.date(2009, 1, 16)'
>>> eval(_)                   # _ is the output of the last command
datetime.date(2009, 1, 16)
```

The output is a string that can be parsed by the python interpreter and results in an equal object.

If that's not possible, it should return a string in the form of `<...some useful description...>` .

edited Nov 25 '09 at 6:39

answered Jan 16 '09 at 22:41


Georg Schölly
64.7k 23 136 205

3 Notice the downside of trying to have your __repr__() produce an expression that returns a copy of the object: you have to assume that the original package was imported completely. If you just run Georg's three lines you'll get "NameError: name 'datetime' is not defined" because the datetime package was never imported. Only "date" from "datetime" was imported. – Michael Scott Cuthbert Apr 12 '13 at 23:27

add a comment

It should be a Python expression that, when eval'd, creates an object with the exact same properties as this one. For example, if you have a `Fraction` class that contains two integers, a numerator and denominator, your `__repr__()` method would look like this:

```
# in the definition of Fraction class
def __repr__(self):
    return "Fraction(%d, %d)" % (self.numerator, self.denominator)
```

Assuming that the constructor takes those two values.

edited Jan 17 '09 at 15:19
J.F. Sebastian
108k 19 178 291

answered Jan 16 '09 at 22:42
Jeremy Ruten
63.1k 16 106 157

add a comment

Guideline: If you can succinctly provide an *exact representation*, **format it as a Python expression** (which implies that it can be both eval'd and copied directly into source code, in the right context). If providing an *inexact representation*, **use `<...>` format**.

There are many possible representations for any value, but the one that's most interesting for Python programmers is an expression that recreates the value. Remember that **those who understand Python are the target audience**—and that's also why inexact representations should include relevant context. Even the default `<XXX object at 0xNNN>`, while almost entirely useless, still provides type, `id()` (to distinguish different objects), and indication that no better representation is available.

edited Jan 3 '10 at 21:01

answered Jan 17 '09 at 14:42
Roger Pate

add a comment

"but does that mean it should just be an example of the sort of expression you could use, or should it be an actual expression, that can be executed (eval etc..) to recreate the object? Or... should it be just a rehasing of the actual expression which was used, for pure information purposes?"

Wow, that's a lot of hand-wringing.

1. An "an example of the sort of expression you could use" would not be a representation of a specific object. That can't be useful or meaningful.
2. What is the difference between "an actual expression, that can ... recreate the object" and "a rehasing of the actual expression which was used [to create the object]"? Both are an expression that creates the object. There's no practical distinction between these. A repr call could produce either a new expression or the original expression. In many cases, they're the same.

Note that this isn't always possible, practical or desirable.

In some cases, you'll notice that repr() presents a string which is clearly *not* an expression of any kind. The default repr() for any class you define isn't useful as an expression.

In some cases, you might have mutual (or circular) references between objects. The repr() of that tangled hierarchy can't make sense.

In many cases, an object is built incrementally via a parser. For example, from XML or JSON or something. What would the repr be? The original XML or JSON? Clearly not, since they're not Python. It could be some Python expression that generated the XML. However, for a gigantic XML document, it might not be possible to write a single Python expression that was the functional equivalent of parsing XML.

answered Jan 17 '09 at 3:16
S.Lott
211k 38 291 560

add a comment
