

Re: Signed zeros: is this a bug?

Re: Signed zeros: is this a bug?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-03/msg01460.html>

- *From:* aleax@xxxxxxx (Alex Martelli)
 - *Date:* Sun, 11 Mar 2007 10:26:57 -0700
-

Mark Dickinson <dickinsm@xxxxxxxx> wrote:

I get the following behaviour on Python 2.5 (OS X 10.4.8 on PowerPC, in case it's relevant.)

```
x, y = 0.0, -0.0
x, y
```

(0.0, 0.0)

```
x, y = -0.0, 0.0
x, y
```

(-0.0, -0.0)

I would have expected `y` to be `-0.0` in the first case, and `0.0` in the second. Should the above be considered a bug, or is Python not expected to honour signs of zeros? I'm working in a situation involving complex arithmetic where branch cuts, and hence signed zeros, are important, and it would be handy if the above code could be relied upon to do the right thing.

Looks to me like you've found a bug with the parser/compiler:

```
c=compile('-0.0,0.0', 'eval', 'eval')
eval(c)
```

(-0.0, -0.0)

```
dis.dis(c)
```

```
1 0 LOAD_CONST 1 ((-0.0, -0.0))
3 RETURN_VALUE
```

Similar problems appear in parsing display forms of lists and dicts and

Re: Signed zeros: is this a bug?

Re: Signed zeros: is this a bug?

calls to functions with constant arguments (among which a constant 0.0 and a constant -0.0) -- and more generally when both constants -0.0 and 0.0 appear within the same "unit of compilation" (expression, function, ...) as for example in:

```
def f():  
  
... a = -0.0  
... return a, 0.0  
...  
  
f()  
  
(-0.0, -0.0)
```

Why I think it has specifically to do with parsing/compiling, e.g.:

```
{23:-0.0, 45:0.0}  
  
{45: -0.0, 23: -0.0}  
  
x=0.0  
y=-0.0  
{23:x, 45:y}  
  
{45: -0.0, 23: 0.0}
```

so there appears to be no problem once the 0.0 and -0.0 values come from variables or the like -- only if they're both from constants within the same "unit of compilation". Indeed, if you put the code above in a function, rather than in separate entries to the toplevel interpreter...:

```
def g():  
  
... x = 0.0  
... y = -0.0  
... return {23:x, 45:y}  
...  
  
g()
```

Re: Signed zeros: is this a bug?

```
{45: 0.0, 23: 0.0}
```

....the bug surfaces again. Looks a bit like the result of a slightly errant "peephole optimizer" pass which tries to roll multiple occurrences of "the same constant" within the same codeobject into a single one, and consider two immutables a "multiple occurrence" if they're == to each other (as, indeed, 0.0 and -0.0 must be).

The Python-coded compiler package does NOT give the problem:

```
compiler.compile('-0.0,0.0','zap','eval')
```

```
<code object <expression> at 0x70068, file "zap", line 1>
```

```
dis.dis(_)
```

```
1 0 LOAD_CONST 1 (0.0)
3 UNARY_NEGATIVE
4 LOAD_CONST 1 (0.0)
7 BUILD_TUPLE 2
10 RETURN_VALUE
```

....presumably because it doesn't do peephole optimization (nor any other kind of optimization).

Unfortunately I've never really looked in depth into these parts of Python so I can't help much; however, if you open a bug at http://sourceforge.net/tracker/?group_id=5470 I think you stand a good chance of eventually seeing it fixed in 2.5.x for some x.

Python/peephole.c does appear to be taking some care in constant folding:

```
192 case UNARY_NEGATIVE:
193 /* Preserve the sign of -0.0 */
194 if (PyObject_IsTrue(v) == 1)
195 newconst = PyNumber_Negative(v);
```

so I suspect the problem is elsewhere, but I can't find it yet.

Alex

In the meantime, I hope that some available workarounds for the bug are clear from this discussion: avoid using multiple constants in a single compilation unit where one is 0.0 and another is -0.0, or, if you really can't avoid that, perhaps use `compiler.compile` to explicitly build the

Re: Signed zeros: is this a bug?

Re: Signed zeros: is this a bug?

bytecode you need.

.