

## Re: Interesting list Validity (True/False)

---

*Source:* <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-05/msg02371.html>

---

- *From:* Steven D'Aprano <[steven@xx](mailto:steven@xx)>
  - *Date:* 16 May 2007 02:23:33 GMT
- 

On Mon, 14 May 2007 11:41:21 -0700, mensanator@xxxxxxx wrote:

On May 13, 8:24 am, Steven D'Aprano  
<[s...@xx](mailto:s...@xx)> wrote:

On Sat, 12 May 2007 21:50:12 -0700, mensana...@xxxxxxx wrote:

I intended to reply to this yesterday, but circumstances (see timeit results) prevented it.

Actually, it's this statement  
that's non-sensical.

```
<quote>  
"if arg==True" tests whether  
the object known as arg is  
equal to  
the object known as True.  
</quote>
```

Not at all, it makes perfect sense.  $X == Y$   
always tests whether the  
argument  $X$  is equal to the object  $Y$   
regardless of what  $X$  and  $Y$  are.

Except for the exceptions, that's why the statement is wrong.

But there are no exceptions.

## Re: Interesting list Validity (True/False)

<quote emphasis added>

Sec 2.2.3:

Objects of different types, \*--->except<---\* different numeric types and different string types, never compare equal; </quote>

Yes, and all swans are white, except for the black swans from Australia, but we're not talking about swans, nor are we talking about objects of different type comparing unequal, we're talking about whether `X == Y` means X is equal to Y.

THERE ARE NO EXCEPTIONS TO THIS, BECAUSE IT IS TRUE BY DEFINITION.

In Python, the meaning of "equal" is nothing more and nothing less than "does `X == Y` return True?". End of story, there is nothing more to discuss. If it returns True, they are equal. If it doesn't, they aren't.

If you want to drag in non-Python meanings of "equal", you are wrong to do so. "Lizzie Windsor", "Queen Elizabeth the Second", "the Queen of England" and "Her Royal Majesty, Queen Elizabeth II" are all equal in the sense that they refer to the same person, but it would be crazy to expect Python to compare those strings equal.

If you want to complain that lists and tokens should compare equal if their contents are the same, that's a different issue. I don't believe you'll have much support for that.

If you want to complain that numeric types shouldn't compare equal, so that `1.0 != 1 != 1L != gmpy.mpz(1)`, that's also a different issue. I believe you'll have even less support for that suggestion.

[snip]

No, they are not "equal".

Of course they are. It says so right there: "a equals d" is true.

Ok, but they are an exception to the rule "different types compare False".

You are only quoting part of the rule. The rule says that numeric types and strings are not included in the "different types" clause. If you quote the full rule, you will see that it is not an exception to the rule, it matches perfectly.

## Re: Interesting list Validity (True/False)

Although, the rule as given is actually incomplete, because it only applies to built-in types. It does not apply to classes, because the class designer has complete control over the behaviour of his class. If the designer wants his class to compare equal to lists on Wednesdays and unequal on other days, he can. (That would be a stupid thing to do, but possible.)

[snip]

The ints  
ALWAYS have to be coerced to mpz to perform arithmetic  
and this  
takes time...LOTS of it.

Really? Just how much time?

Can't say, had to abort the following. Returns the count of  $n/2$  and  $3n+1$  operations [1531812, 854697].

Maybe you should use a test function that isn't so insane then. Honestly, if you want to time something, time something that actually completes! You don't gain any accuracy by running a program for twenty hours instead of twenty minutes.

[snip functions generating the Collatz sequence]

```
timeit.Timer("x == y", "import gmpy; x = 1; y = gmpy.mpz(1)").repeat()  
timeit.Timer("x == y", "x = 1; y = 1").repeat()
```

I don't have gmpy installed here,

Good Lord! How do you solve a Linear Congruence? :-)

In my head of course. Don't you?

\*wink\*

so I can't time it, but I look forward to seeing the results, if you would be so kind.

## Re: Interesting list Validity (True/False)

I had a lot of trouble with this, but I think I finally got a handle on it. I had to abort the previous test after 20+ hours and abort a second test (once I figured out to do your example) on another machine after 14+ hours. I had forgotten just how significant the difference is.

```
import timeit
```

```
## t = timeit.Timer("a == b", "a = 1; b = 1") ## u =  
timeit.Timer("c == d", "import gmpy; c = 1; d = gmpy.mpz(1)")  
## t.repeat()  
## [0.22317417437132372, 0.22519314605627253, 0.22474588250741367] ##  
u.repeat()  
## [0.59943819675405763, 0.5962260566636246, 0.60122920650529466]
```

Comparisons between ints take about 0.2 microseconds, compared to about 0.6 microseconds for small gmpy.mpz values. That's an optimization worth considering, but certainly not justifying your claim that one should NEVER compare an int and a mpz "in a loop". If the rest of the loop takes five milliseconds, who cares about a fraction of a microsecond difference?

Unfortunately, this is not a very useful test, since mpz coercion appears to vary by the size of the number involved.

No, it is a very useful test. It's not an EXHAUSTIVE test.

(By the way, you're not testing coercion. You're testing the time it takes to compare the two. There may or may not be any coercion involved.)

Although changing t to

```
## t = timeit.Timer("a == b", "a = 2**177149-1; b = 2**177149-1")  
  
still produces tractable results  
## t.repeat()  
## [36.323597552202841, 34.727026758987506, 34.574566320579862]
```

About 36 microseconds per comparison, for rather large longints.

the same can't be said for mpz coercion:

Re: Interesting list Validity (True/False)

```
## u = timeit.Timer("c == d", "import gmpy; c = 2**177149-1; d =  
gmpy.mpz(2**177149-1)")  
## u.repeat()  
## *ABORTED after 14 hours*
```

This tells us that a comparison between large longints and large gmpz.mpz vales take a minimum of 14 hours divided by three million, or roughly 17 milliseconds each. That's horribly expensive if you have a lot of them.

It isn't clear `_why_` the comparison takes so long.

[snip]

And, just for laughs, I compared mpzs to mpzs,

```
s = 'import gmpy; a = gmpy.mpz(%d); b = gmpy.mpz(%d)' % (n,n)
```

which ended up faster than comparing ints to ints.

I'm hardly surprised. If speed is critical, gmpy is likely to be faster than anything you can do in pure Python.

[snip]

Even if it is terribly slow, that's just an implementation detail. What happens when Python 2.7 comes out (or Python 3.0 or Python 99.78) and coercion from int to mpz is lightning fast? Would you then say "Well, int(1) and mpz(1) used to be unequal, but now they are equal?"

Are you saying I should be unconcerned about implementation details? That it's silly of me to be concerned about implementation side effects due to mis-matched types?

Of course not. But the discussion isn't about optimization, that's just an irrelevant side-track.

Me, I'd say they always were equal, but previously it used to be slow to coerce one to the other.

## Re: Interesting list Validity (True/False)

So, when you're giving advice to the OP you don't feel any need to point this out? That's all I'm trying to do, supply some "yes, but you should be aware of..." commentary.

Why on Earth would I need to mention `gmpy.mpz()`? Does the OP even use `gmpy`? You were the one who brought `gmpy` into the discussion, not him. Why not give him a lecture about not repeatedly adding strings together, or using `<<` instead of multiplication by two, or any other completely irrelevant optimization? My favorite, by the way, is that you can save anything up to an hour of driving time by avoiding Hoddle Street during peak hour and using the back-streets through Abbotsford, next to Yarra Bend Park and going under the Eastern Freeway. Perhaps I should have raised that as well?

In any case, what you describe is a local optimization. Its probably a good optimization, but in no way, shape or form does it imply that `mpz(1)` is not equal to 1.

It's a different type. It is an exception to the "different types compare False" rule.

What does this have to do with your ridiculous claim that `mpz(1)` is not equal to 1? It clearly is equal.

That exception is not without cost, the type mis-match causes coercion.

Any comparison has a cost. Sometimes its a lot, sometimes a little. That has nothing to do with equality.

There's nothing false about it. Ask any mathematician, does 1 equal 1.0, and they will say "of course".

And if you ask any mathematician, he'll say that (1,) is equal to [1].

I'd like to find the mathematician who says that. The first thing he'd say is "what is this (1,) notation you are using?" and the second thing

Re: Interesting list Validity (True/False)

he'd ask is "equal in what sense?".

Perhaps you should ask a mathematician if the set  $\{1, 2\}$  and the vector  $[1, 2]$  are equal, and if either of them are equal to the coordinate pair  $(1, 2)$ .

That's the difference between a mathematician and a programmer. A programmer will say "of course not, the int has to be coerced."

A C programmer maybe.

[snip]

Numeric values are automatically coerced because that's more practical. That's a design decision, and it works well.

And I'm not saying it shouldn't be that way. But when I wrote my Collatz Functions library, I wasn't aware of the performance issues when doing millions of loop cycles with numbers having millions of digits. I only found that out later. Would I have gotten a proper answer on this newgroup had I asked here? Sure doesn't look like it.

If you had asked `_what_`? Unless you tell me what question you asked, how can anyone guess what answer you would have received?

If you had asked a question about optimization, you surely would have received an answer about optimization.

If you asked about string concatenation, you would have received a question about string concatenation.

If you had asked a question about inheritance, you would have received an answer about inheritance.

See the pattern?

BTW, in reviewing my Collatz Functions library, I noticed a coercion I had overlooked, so as a result of this discussion, my library is now slightly faster. So some good comes out of this argument after all.

## Re: Interesting list Validity (True/False)

As for gmpy.mpz, since equality tests are completely under the control of the class author, the gmpy authors obviously wanted mpz values to compare equal with ints.

And they chose to do a silent coercion rather than raise a type exception.

It says right in the gmpy documentation that this coercion will be performed.

What it DOESN'T say is what the implications of this silent coercion are.

OF COURSE a coercion takes time. This is Python, where everything is a rich object, not some other language where a coercion merely tells the compiler to consider bytes to be some other type. If you need your hand-held to the point that you need somebody to tell you that operations take time, maybe you need to think about changing professions.

The right way to do this is to measure first, then worry about optimizations. The wrong way is to try to guess the bottlenecks ahead of time. The worse way is to expect other people to tell you where your bottlenecks are ahead of time.

Since both lists and tuples are containers, neither are strings or numeric types, so the earlier rule applies: they are different types, so they can't be equal.

But you can't trust `a==d` returning True to mean a and d are "equal".

What does it mean then?

It means they are mathematically equivalent, which is not the same as being programatically equivalent. Mathematical equivalency is what most people want most of the time.

I think that by "most people", you mean you.

Re: Interesting list Validity (True/False)

## Re: Interesting list Validity (True/False)

Not all of the people all of the time,  
however. For example, I can calculate my Hailstone Function parameters  
using either a list or a tuple:

```
import collatz_functions as cf
print cf.calc_xyz([1,2])
```

```
(mpz(8), mpz(9), mpz(5))
```

```
print cf.calc_xyz((1,2))
```

```
(mpz(8), mpz(9), mpz(5))
```

But `[1,2]==(1,2)` yields `False`, so although they are not equal, they ARE interchangeable in this application because they are mathematically equivalent.

No, they aren't mathematically equivalent, because Python data structures aren't mathematical entities. (They may be \_similar to\_ mathematical entities, but they aren't the same. Just ask a mathematician about the difference between a Real number and a float.)

They are, however, both sequences, and so if your function expects any sequence, they will both work.

[snip]

I never said that there was no efficiency differences. Comparing X with Y might take 0.02ms or it could take 2ms depending on how much work needs to be done. I just don't understand why you think that has a bearing on whether they are equal or not.

The bearing it has matters when you're writing a function library that you want to execute efficiently.

Which is true, but entirely irrelevant to the question in hand, which is "are they equal?".

--

Steven.

Re: Interesting list Validity (True/False)

Re: Interesting list Validity (True/False)