

Re: Some more notes

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2004-10/3883.html>

From: Josiah Carlson (jcarlson_at_uci.edu)

Date: 10/23/04

Date: Sat, 23 Oct 2004 11:55:15 -0700

To: bearophile@lycos.com (bearophile), python-list@python.org

bearophileHUGS@lycos.com (bearophile) wrote:

> *Cliff Wells:*

>

> > *Actually, since the assignment operator is used far more often than
> the*

> > *equality operator, making it require three keystroke rather than just
> > one is a bad idea.*

>

> > *Right, still, from a mathematical/formal point of view, Pascal syntax
> is better here :-]*

Based on Python's use of '=' rather than ':=', I believe Guido has already made his decision, and there has been no discussion on changing the behavior for Python 3.0, so argument over it is a moot point.

> > *Occasionally useful. Not enough to put it high on my list of things*

> > *I'd*

> > *like to see in Python, but I can see its utility for making*

> > **slightly**

> > *more logical code (versus the "while True: if x: break" idiom).*

>

> > *I agree, it's not essential, but it can be useful :-)*

As it stands, there has been more than one attempt to get case statements into the Python language. All have been mostly fruitless thusfar. There is a PEP: <http://python.org/peps/pep-0275.html>, but it is certainly not going to make it into Python 2.4, and there has been little discussion about its inclusion in any later versions on python-dev.

In my opinion, the syntax-less case statement described in the PEP is pure. That is, no new syntax is needed, but if your if/elif/else statements are of a certain format, you gain the speed of dictionary dispatch. I believe that any case-statement-like behavior in Python will likely be of this sort.

- > >Why not just use "/"?
- >
- > Okay, I'll use `os.path.normcase` to convert them.

Not necessary. You can mix '/' and '\\' freely on Windows when opening files.

- > >but there are plenty of things to confuse newbies,
- >
- > *Reducing non-obvious, magic or not-standard behaviours is quite*
- > *necessary inside a language, it helps non-newbies too, making the*
- > *language more conceptually transparent. Sometimes such non-standard*
- > *behaviours can be useful to improve language speed or other things,*
- > *but they have to be reduced to the minimum possible. Python often prefers*
- > *transparency, even when there are many (worse and more opaque, but*
- > *maybe faster) ways to do things. This is positive for a hi-level*
- > *language. So making a hi-level language like Python more transparent*
- > *can be a way to improve it :-)*

What you just said was that Python is transparent, but making it more transparent is better. What would be more transparent? Understand that Python is not likely to do anything that would make it more difficult to learn or use, just for the sake of speed.

- > *Mathematica is more transparent than Python (its hashes can process*
- > *everything, it has just a mutable list type and not tuples, assignment*
- > *copies the object, and it's more functional so there are much less*
- > *problems with globals and side effects, etc. but it can still be used*
- > *with imperative programming) and often it's also faster, but it's far*
- > *from free (and it can be worse to do "real" programming, etc) :-]*

It would seem that copy-on-write for lists was a work-around for being able to hash sequences, a result of only having mutable lists (no immutable lists/tuples).

I don't remember copy-on-write with Mathematica, though it has been a few years, I could have sworn that either append or prepend was fast (which necessitates non-copy-on-write semantics).

"problems with globals and side effects"

The only problem is education. Once one becomes educated about the side-effects of append/extend/etc., one rarely has problems of this kind, as one either uses them to their advantage, or programs around what they conceive to be a limitation. Feel free to program around them; I'm going to stick with using mutables wherever I see fit. Please stop advocating the removal of useful features.

"often it's also faster"

Honestly, I (and likely many others) don't care that Mathematica can be faster. C can be faster, Java can be faster, Perl can be faster. We use Python for varying reasons and for varying purposes.

comp.lang.python: Re: Some more notes

In my case, the only thing I find Mathematica useful for is helping with certain difficult bits of math that I have forgotten over the years.

For general programming, there are other better languages (in my opinion, which I imagine is shared), nearly all of which are free.

> *Note: Mathematica can be used with a quite powerful "pattern matching programming", I don't know if such thing (or a subset of it) can be useful to add to Python.*

If you mean things like (I can't remember the exact syntax, perhaps this is it)...

```
fib[a_] := If[a<3, Return[1], b=fib[a-1]+fib[a-2];fib[a]=b;Return[b]]
```

That does what is called memoization in computer science. Python can do that, but you need to do so explicitly:

```
def fib(a, cache={}):
    if a < 3:
        return 1
    if a not in cache:
        cache[a] = fib(a-1)+fib(a-2)
    return cache[a]
```

Ahh, dictionaries save the day. With a 'memoization decorator', it gets easier in the general case

```
def memoize_single_arg(f)
    cache = {}
    def memoizer(arg):
        if arg in cache:
            return cache[arg]
        return f(arg)
    return memoizer
```

```
@memoize_single_arg
def fib_r(a):
    if a < 3:
        return 1
    return fib_r(a-1)+fib(a-2)
```

If you are talking about something else, perhaps you should describe it.

> *>there is a logical explanation that suffices in
> >place of breaking tons of Python code.
>
> There are logical explanations for everything (because everything must
> be executed by the processor), but for humans some things are more
> logical than others :-)
> Python 3.0 already breaks lots of things, even division operators, so
> that's the best occasion to improve/change/fix things.*

Funny, I found no mention of division operator breaking on the Python 3.0 wiki: <http://www.python.org/cgi-bin/moinmoin/PythonThreeDotOh>

Python 3.0 doesn't even exist yet, and likely won't for at least 6 years. Virtually none of the suggestions you have offered thus far are even remotely what Guido and others on python-dev have been talking about changing for Python 3.0

> *Jeff Shannon:*
>
> *>using a dict of functions is (IMO) a better and cleaner*
> *>way of implementing the same idea.*
>
> *I don't know... The Case syntax can be something like:*
>
> *Case <name>:*
> *1: DoSomething1*
> *range(2,23): DoSomething2*
> *else: DoSomething3*

That is the worst syntax for case statements I have ever seen.

> *>Except that having mutable types be shared this way can be a useful*
> *feature.<*
>
> *It looks more dangerous than useful. And assignments of non-mutable*
> *types can still be accepted.*

Understand that 'obj.attr = val' implies that obj is mutable. Do you also want to remove object oriented programming in Python? Likely not, but understand the implications for what you say.

Mutables aren't going away in Python. Stop complaining about them.

> *>Having a += x mean in-place modification*
> *>while a = a + x creates a new object allows both options to be*
> *>easily accessible without function-call syntax.*
>
> *Okay, the manual I've read wasn't clear enough about this :-)*

It is in the language reference:
<http://docs.python.org/ref/augassign.html>

"An augmented assignment expression like `x += 1` can be rewritten as `x = x + 1` to achieve a similar, but not exactly equal effect. In the augmented version, `x` is only evaluated once. Also, when possible, the actual operation is performed in-place, meaning that rather than creating a new object and assigning that to the target, the old object is modified instead."

– Josiah