

Re: any macro-like construct/technique/trick?

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2005-06/msg00230.html>

- *From:* "Jordan Rastrick" <jrastrick@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* 1 Jun 2005 18:34:49 -0700
-

Is having to read two lines really that much worse than one? And the only thing you find objectionable about the most obvious solution? If so, then what's wrong with:

```
def foo():
# .... do some stuff
if debug: emit_dbg_obj(DbgObjFoo(a,b,c))
# .... do more stuff
```

To my mind, this is no less readable and no more clutter than `DEBUG_EMIT(DbgObjFoo(a,b,c))` Then again, I don't mind the two line version – personally, I prefer my debug code be more conspicuous rather than less, so that its clearly separate, visually and hence mentally, from functional part of the code.

Preprocessor Macros and their ilk have always seemed like an awful kludge to me, nessecary and useful in a language like C I'll grant, but not very Pythonic at all.

If you really feel you can't stand the (presumably marginal) expense of creating the debug objects when unnessecary, and want to get rid of certain lines of code conditionally – using textual replacement, which is essentially, by my limited understanding, all a pre-processor does – then why not just do it yourself? Have `emit_dbg_obj(DbgObjFoo(a,b,c))` all you want in your in development code, and use a simple regex search and replace to comment it out of your source when the code ships.

Most critically, though, if you're worried about efficiency issues this fine-grained, trying to scrounge every last byte of memory, I'd suggest Python is probably the wrong language for your problem. Object creation overhead and the like is just part of the regular cost of having a high level, dynamic, interpreted language. If you really, really need preprocessor macros for efficiency reasons, then your probably need C.

I'm sorry, my reply hasn't been all that helpful I guess :) It seems a bit of a trend on this mailing list – ask for some feature, or a way to emulate it, and instead get a bunch of arguments as to why that feature doesn't fit the design and philosophy of the language, and some

Re: any macro-like construct/technique/trick?

suggestions on how to do things differently.

Probably some much more knowledgeable Pythonista can cook up a way to achieve your desired behaviour, using `exec` or `eval` or some other such unsightly hack... but I can only hope such practices don't become widespread.

Mac wrote:

```
> Is there a way to mimic the behaviour of C/C++'s preprocessor for
> macros? The problem: a lot of code like this:
>
> def foo():
> # .... do some stuff
> if debug:
> emit_dbg_obj(DbgObjFoo(a,b,c))
>
> # .... do more stuff
> if debug:
> emit_dbg_obj(DbgObjBar(d,e))
>
> # ... and so on ...
>
> Notes:
>
> * the two-lines of debug conditional tend to really break up the flow
> of the surrounding code
>
> * in C you could wrap them with a macro so you could do
> DEBUG_EMIT(DbgObjFoo(a,b,c)), etc, with the macro only instantiating
> the object and processing it if the debug flag was set. The one-liner
> is MUCH less disruptive visually when reading code
>
> * using
> def debug_emit(obj):
> if debug:
> emit_dbg_obj(obj)
> is a poor solution, because it *always* instantiates DbgObj*, even when
> not needed; I want to avoid such unnecessary waste
```

- **Follow-Ups:**

- ◆ **Re: any macro-like construct/technique/trick?**

- ◇ From: Mac

- **References:**

Re: any macro-like construct/technique/trick?

◆ *any macro-like construct/technique/trick?*

◇ *From:* Mac

- Prev by Date: *Re: Performance Issues please help*
- Next by Date: *Re: Strange KeyError using cPickle*
- Previous by thread: *any macro-like construct/technique/trick?*
- Next by thread: *Re: any macro-like construct/technique/trick?*
- Index(es):
 - ◆ *Date*
 - ◆ *Thread*