

Re: About those parenthesis....

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2006-01/msg00694.html>

- *From:* "Kaz Kylheku" <kkylheku@xxxxxxxxx>
 - *Date:* 10 Jan 2006 23:35:56 -0800
-

Majorinc wrote:

> I like parenthesis and prefix from beginning. However, I think
> that usual $f(x,y,z)$ would be better choice than $(f\ x\ y\ z)$.

The commas don't serve any purpose, other than to add clutter. So a better notation than $f(x, y, z)$ is quite simply $f(x\ y\ z)$.

I have programmed with Lisp-like semantics using the $f(x,y,z)$ syntax. It's ugly. The comas really are a problem. There is no satisfactory way to format them. The syntax does not scale to argument lengths required in Lisp programming, where long expressions have to be split across lines.

The commas have to go. At that point, it makes sense to "castle" the function by exchanging it with the parenthesis.

Hanging on the outside of the parenthesis, the function isn't syntactically encapsulated. There is an ambiguity there. A left-to-right scan, having seen only the identifier f , cannot decide whether f is a plain term, or an operation with arguments. The scan must continue to the next token before that can be decided. Until then, the scanner is in a state that is a combination of two states: "I am parsing either a simple term, or a the beginning of a compound". If you put the parenthesis first, it's clear. You read a token. It's a parenthesis: great, we have a compound. It's an identifier, so we have a term.

The $f(x\ y\ z)$ notation has no nice way to denote the empty list, because the token f is needed for syntax. The notation $()$ is a syntax error because it has no operator on the left.

The $f(x\ y\ z)$ notation is also very poor when it comes to subexpressions which do not represent an operation f being applied to arguments $x\ y\ z$. It assumes certain semantics: namely that the left item is elevated above the others. This is not the case when you have syntactic abstraction.

Consider what happens to this:

Re: About those parenthesis....

```
(defclass c (b1 b2 b3) ())
```

Let's skip the fact that we don't know what to do with the empty list of slot definitions (). In fact, let's invent the @() notation for that, what the hell. So we get:

```
defclass (c b1 (b2 b3) @())
```

Do you see how wrong that is? (b1 b2 b3) is just a list of base classes. Multiple inheritance from three ancestors. This is mangled into b1 (b2 b3) which doesn't make sense at all. b1 isn't an operator, it's just a base class like b2 and b3.

Even when the first symbol /is/ an operator, it's not more important than the other two symbols.

When you write (+ 1 3) to denote 4, the 4 is a result of all three symbols acting together. Each one contributes as much to the result as the other two. The + does not deserve any more credit than the 3 for that result. It's the combination of these three symbols in that order, and the surrounding broader context, that gives rise to 4.

So, in summary, f(x, y, z) is an abomination that rightfully deserves to be cast into the lake of fire.

.

- *Follow-Ups:*

- ◆ [Re: About those parenthesis....](#)

- ◆ *From:* Marcin 'Qrczak' Kowalczyk

- *References:*

- ◆ [About those parenthesis....](#)

- ◆ *From:* Joe Marshall

- ◆ [Re: About those parenthesis....](#)

- ◆ *From:* Majorinc , Kazimir

- Prev by Date: [Re: About those parenthesis....](#)
- Next by Date: [Re: About those parenthesis....](#)
- Previous by thread: [Re: About those parenthesis....](#)
- Next by thread: [Re: About those parenthesis....](#)
- Index(es):
 - ◆ [Date](#)
 - ◆ [Thread](#)