

Re: Difference between ' and : symbols

Source: <http://coding.derkeiler.com/Archive/Lisp/comp.lang.lisp/2005-07/msg00415.html>

- *From:* cstacy@xxxxxxxxxxxxxx (Christopher C. Stacy)
 - *Date:* Fri, 08 Jul 2005 08:07:55 GMT
-

"jonathon" <j_mckitrick@xxxxxxxxxxxx> writes:

> I've noticed that with ASDF there are 2 ways of referring to a
> system. Some use the single quote for the name, others use the
> colon before the symbol. Both seem to work.

>

> What is the difference? Isn't a symbol evaluated to itself,
> while a single-quoted value is evaluated to that value?

Symbols do *_not_* evaluate to themselves; they evaluate to the value of the variable (if any) that they are naming.

Let's start out by reviewing how evaluation works.

```
lisp> x
=> {error: unbound variable X}
lisp> (setq x 12345)
lisp> x
=> 12345
```

Normally, all the arguments to a function are recursively evaluated before the function is called on them.

```
lisp> (defun foo (a b)
(list a b))
```

```
lisp> (foo x (* x 2))
=> (12345 24690)
```

You can inhibit evaluation with the special operator QUOTE. QUOTE looks like a function call, but it's not!

```
lisp> (quote hello)
=> HELLO
```

QUOTE is a "special operator". What's special is that it does not obey the normal rule for Lisp evaluation. QUOTE does not evaluate its arguments; instead, it returns the arguments literally.

Re: Difference between ' and : symbols

You can abbreviate QUOTE as the quote character (').

```
lisp> 'hello  
=> HELLO
```

```
lisp> '(* x 2)  
=> (* X 2)
```

Notice that in contrast to symbols, numbers (eg. 12345) normally evaluate to themselves. So do strings.

```
lisp> 12345  
=> 12345
```

```
lisp> "hello, world"  
=> "hello, world"
```

By the way, if you quote a number or a string, it's totally gratuitous, but you'll end up with the same result:

```
lisp> '12345  
=> 12345
```

Numbers and strings are "self-evaluating".
Symbols are not self-evaluating.

```
lisp> x  
=> 12345
```

Symbols only "evaluate to themselves" if you use QUOTE. Otherwise, symbols evaluate to their variable value.

QUOTE breaks out of the normal evaluation rules, and always makes its argument "evaluate to itself". It's a way to introduce a literal value (for a symbol or a list, which normally would be evaluated, and thus otherwise would have turned into the resulting value of a variable or a function call).

Before we proceed, there is another detail about symbols that you ought to understand.

Every symbol is an object which has several attributes. A symbol has a name, such as X, and the symbol might also have a variable value binding, such as 12345.

A symbol also has an attribute called its "package", which is part of its fully qualified name. The syntax of a symbol's name is PACKAGE:NAME.

Beginners usually take no notice of the package.

Re: Difference between ' and : symbols

Re: Difference between ' and : symbols

This is because we usually just see the abbreviated "name" part of the symbol's name. Lisp doesn't usually need to print out the symbol's fully qualified name, which would include the package prefix.

When you start up Lisp, the current package (analogous to the idea of a current working directory in a file system) is called "CL-USER". Any symbols that you make up will be inside of ("interned in") this package. When you type FOO, the Lisp READ function that is processing your input knows that this meant CL-USER:FOO. And the PRINT function, likewise, doesn't bother to print out the package prefix. It simply prints the symbol's name as FOO.

(Let's never mind under what circumstances the fully qualified name of the symbol would be printed, or when you might have to input the package name. It has to do with disambiguating the symbol in the presence of more than one package, and something with having the current package set to something different than the symbol's package. It also has to do with whether the symbol has been imported/exported from the package as an external symbol. None of those details are relevant to your question.)

We can now come to explain the confusing things from your experience with ASDF.

There is a special package named KEYWORD. Symbols in the KEYWORD package are called "keywords". Keywords are a kludge in the language:

* Symbols do not evaluate to themselves.
Except for keywords!

```
lisp> x
=> 12345
lisp> keyword:x
=> :X
lisp> :x
=> :X
```

As you can see, keywords are magic, and somewhat unlike ordinary symbols.

Notice that the package name, KEYWORD, is not normally printed, nor does it need to be entered. To indicate the keyword package, leave off "KEYWORD" and just use the delimiter (":") that would form the prefix.

Notice that the value of X and :X are not the same. That's because X and :X are two distinct objects.

Re: Difference between ' and : symbols

They are both symbols, and they happen to have the same name as each other, but they are in different packages.

Keywords are a convenient kludge in the language that allows you to easily input a symbol, useful for certain purposes, that will never have a variable value. Rather, it will always "evaluate to itself" (that is, its name, constantly).

Lisp "keywords" are employed for much the same purpose that keywords serve in other programming languages. To wit, they are used as unambiguous syntactic markers. Many functions can take arguments specified by keyword.

```
lisp> (position #\C "aAbBcCdDC" :test #'char=)
=> 5
lisp> (position #\C "aAbBcCdDC" :test #'char= :from-end t)
=> 8
```

Keywords are "self-evaluating", thus like numbers or strings, but unlike normal symbols in that regard.

The specific thing that confused you is why ASDF functions accept either symbols or keywords as system names.

The general answer is that some programs work by maintaining some sort of data table that maps one object to another object for the purposes of the application.

Each symbol is a unique and distinct object, and could therefore be used as the lookup key in such a table. In source code, in order to refer to the symbol itself, and not its variable value, you must QUOTE it. (Otherwise, due to the rules of evaluation, you would be referring to whatever variable value the symbol might have.)

A keyword is a unique and distinct object, and could therefore be used as the lookup key in such a table. You don't need to quote keywords. One of the essential features of keywords is that you don't have to quote them. And they always evaluate to the same thing: themselves.

You could quote your keywords (or numbers or strings) if you wanted to. It does not change their value.

```
lisp> (quote 12345)
=> 12345
lisp> :x
=> :X
lisp> (quote :x)
=> :X
```

Re: Difference between ' and : symbols

So why would anyone ever bother to quote a keyword?
They might do that if they were trying to emphasize to someone that they were using the keyword as a piece of data, as opposed to it being a syntactic marker in some function call that accepts keyword-specified arguments.

```
lisp> (find-solution previous-run :method ':alpha-beta)
=> nil
```

```
lisp> (prune-directory "/My" :keep ':oldest)
Pruning all but the oldest files...12 kept, 69 deleted!
```

You still have one last confusing issue, specific to ASDF, and this is where the real confusion comes!

We have learned that in table lookups, generally perhaps one object is as good as another for the purpose of serving as a unique identifier. The ASDF functions want you to pick some identifier to be a "system name"

ASDF seems to accept either a keyword like :FOO, or a symbol (eg. in the default CL-USER package) such as FOO. If you experiment some more, you might discover that ASDF will also accept strings, such as "FOO".

```
lisp> foo
=> {error: unbound variable FOO}
lisp> 'foo
=> FOO
lisp> ':FOO
=> :FOO
lisp> :foo
=> :FOO
lisp> (equal 'foo :foo)
=> NIL
lisp> (equal 'foo "foo")
=> NIL
```

Notice that FOO and :FOO are distinct symbols. You could decide to use one or the other, but they should not be interchangeable, since they are not the same object!

It seems as though you could use ASDF to manipulate three different systems, one named FOO, one named :FOO, and another named "FOO".

The confusing thing is that ASDF accepts any of those identifiers, and somehow treats them all as referring to the same thing. Clearly "FOO" and :FOO are not the same object — they are not even of the same type!

Re: Difference between ' and : symbols

If you look at the source code for ASDF, you will learn that ASDF accepts any of those three pieces of data as a system identifier. But what ASDF really wants is a string.

Every symbol object has a name, which is a string. When given a symbol, ASDF extracts that string from the symbol and just uses the string itself.

Moreover, ASDF wants these strings to be lowercase.

(Symbol names are normally uppercase.)

Whether you handed ASDF a string, or whether ASDF extracted the string from a symbol's name, it calls `STRING-DOWNCASE` on it.

That lowercase string becomes the actual name of the system.

ASDF is generous in accepting either that lowercase string, an upper or mixed-case version of the string, or a symbol, or a keyword, and in any event canonicalizes your input in order to form the actual system name.

If you made it through all that explanation, I think that should probably answer your question.

I'm going to finish this up here by responding to someone else's posting where they tried to explain the mystery to you, but maybe introduced some additional confusion.

This will also help complete the puzzle.

grue@xxxxxxx (Timofei Shatrov) wrote:

```
> Thus, (in-package :asdf) works,  
> but (in-package 'asdf) doesn't  
> because > (quote asdf) is not a string, symbol or character.
```

The above has nothing to do with why ASDF treats two different symbols (`FOO` and `:FOO`) as the same identifier.

Rather, ASDF is massaging the value that you hand it, transforming it into another piece of data.

ASDF is programmed to accept several different kinds of objects as system identifiers, and it turns all of them into the same thing.

I don't see how the above was supposed to answer the ASDF question; I think the poster may be confused. In the example above, with `IN-PACKAGE`, something else entirely is going on.

`IN-PACKAGE` is a macro, not a function.

In Lisp, there are two kinds of operators that can be named as the first element in an expression. The normal thing is a function. Expressions `(F x)` where `F` names a function follow the normal, rules for recursive evaluation -- all the arguments are evaluated, then the function named `F` is called.

Re: Difference between ' and : symbols

But expressions of the form (M x) where M names a macro follow some different rules of evaluation. Macro calls look about the same as function calls, but are special.

Closely related to "macros" are "special forms". They are conceptually similar to each other in that they both resemble function calls, and they both introduce new and arbitrary syntax and evaluation rules.

One special form that you know about already is QUOTE. It does not evaluate its argument, but just returns it literally.

```
lisp> (hello world)
=> {error: unbound variable WORLD}
Maybe undefined function HELLO, too,
but we didn't even get that far.
```

```
lisp> '(hello world)
=> (HELLO WORLD)
```

IN-PACKAGE is not a function. It is a macro, and is defined to process its arguments according to its own idiosyncratic rules. As it happens, it does `_not_` evaluate its arguments.

IN-PACKAGE accepts what it terms a "name", which it defines as a "string designator". If you look up "string designator" in the CLHS Glossary, you will see that this might either be a string, or it might be a symbol.

As documented, IN-PACKAGE does not evaluate this string designator. You can `_not_` use some expression there which would `_evaluate_` to a string or a symbol. The argument must literally `_already_` be a string or a symbol.

Therefore, to set the current package to CL-USER, you could write any of the following:

```
(in-package "CL-USER")
Because that's a string.
Notice that package names are always upper-case.
```

```
(in-package CL-USER)
Because that's a symbol, and IN-PACKAGE will look up
the name of the symbol, which is "CL-USER".
```

```
(in-package :CL-USER)
Because that's a symbol, and IN-PACKAGE will look up
the name of the symbol, which is "CL-USER".
```

Re: Difference between ' and : symbols

(I hope that example wasn't too confusing!
The symbol CL-USER has a package, which is not mentioned here and which is irrelevant. IN-PACKAGE is only interested in extracting the "name" of the symbol, not its package. And then we are using the resulting string for the purpose of manipulating Lisp's "current package".)

You cannot write this:
(in-package 'CL-USER)

Because that argument is just shorthand for this:
(in-package (QUOTE CL-USER))

The transformation of 'CL-USER into (QUOTE CL-USER) is not one of evaluation. It is performed at read-time, when the source code is translated from characters into Lisp objects. (In the typical sequence of events, read-time precedes evaluation.) And in this case, there will be no evaluation of that argument, because IN-PACKAGE is defined to never evaluate its arguments. Plainly, a list is not a string or a symbol, and so this will not be valid.

The only similarity between the ASDF functions and IN-PACKAGE is that they both will accept either strings or symbols. But how they process their arguments, and how they use those arguments internally, are different.

Besides functions like OPERATE (aka OOS), ASDF also does have some macros, notably DEFSYSTEM. Unlike functions, the DEFSYSTEM macro is similar to IN-PACKAGE in that it doesn't evaluate its arguments.

I hope that clears a few things up!

Chris

.

• *Follow-Ups:*

- ◆ *Re: Difference between ' and : symbols*
 ◇ *From:* Tim X
- ◆ *Re: Difference between ' and : symbols*
 ◇ *From:* John
- ◆ *Re: Difference between ' and : symbols*
 ◇ *From:* jonathon

• *References:*

- ◆ *Difference between ' and : symbols*
 ◇ *From:* jonathon

- Prev by Date: *Re: Is semaphore signaling in OpenMCL Threadsafe?*

Re: Difference between ' and : symbols

- Next by Date: ***Re: How to create a standalone GNU/Linux binary using SBCL?***
- Previous by thread: ***Re: Difference between ' and : symbols***
- Next by thread: ***Re: Difference between ' and : symbols***
- Index(es):
 - ◆ ***Date***
 - ◆ ***Thread***