Lisp / Scheme

Alex S.*

1 Lisp

In the late 1950s, John McCarthy (the same one who coined the term "Artificial Intelligence"), created a programming language named Lisp—which stands for "list processing" (it also stands for "Lots of Irritating Superfluous Parentheses").

1.1 Scheme

Modern Lisp, or "Common Lisp" is a rich and bloated language with all sorts of thingies. Scheme is a rather simplified subset of the language. Most things we discuss here apply to both Scheme and Lisp.

There's a yet simpler Lisp subset called 'Stutter'; which is a great place to start learning Lisp.

2 The Language

In Lisp, you program by writing expressions. An exrepssion is either a *list* or an *atom*. An atom is simply a string, or a number (like `glah`, `blah`, `123`). A list is closed by parenthesis, ie: `(a b c)`. An empty list is written as `()` or `nil`.

A function call is represented as a list. `(f b c)` is a function `f` being called with arguments `a` and `b`.

2.1 Interpreter

You can just type an expression and have it evaluated, ie:

```
> 'blah
BLAH
> 'glah
GLAH
```

*alex@theparticle.com

```
> (+ 5 7)
12
> (set 'glah 'blah)
BLAH
> glah
BLAH
> (set 'ten '10)
10
> (* 3.14 ten)
31.400002
```

2.2 Lists

You can operate on lists via the `car` and `cdr` functions. The `car` returns the first element of the list, while `cdr` returns the tail. For example:

```
> (car '(a b c))
A
> (cdr '(a b c))
(B C)
```

You can of course have lists that are sublists of other lists. For example:

```
> (car '((a b c) (d e f) (g h i)))
(A B C)
> (cdr '((a b c) (d e f) (g h i)))
((D E F) (G H I))
```

Another very useful function is to construct a list out of the head and tail. `cons` does that.

```
> (cons 'a nil)
(A)
> (cons 'a '(b c))
(A B C)
> (cons '(A B C) '((D E F) (G H I)))
((A B C) (D E F) (G H I))
```

2.3 Conditionals

There is a familiar `if` function, that works like this:

```
(if 'condition 'if-true-expression 'else-expression)
```

For example:

```
> (set 'pi 3.14)
3.14
> (set 'b (* 2 pi))
6.28
> (if (< pi b) 'pi-less-than 'pi-greater-than)
PI-LESS-THAN
```

2.4 Lambda Expressions

Lambda expressions are basically functions without a name. The form is:

```
(lambda (arg1 arg2) (function body))
```

In other words, a function that increments a value might look like this:

```
(lambda (x) (+ x 1))
```

To try it out, we just need to 'use it', ie:

```
> ((lambda (x) (+ x 1)) 12345)
12346
```

2.5 Functions

You can define a function via `defun`.

```
> (defun increment (x) (+ x 1))
INCREMENT
> (increment 123)
124
```

3 Examples

3.1 Factorial

Factorial can be written like this:

```
(defun factorial (n)
  (if (<= n 1)
      1
    (* n (factorial (- n 1)))))
```

The cool thing about Lisp is that all arithmetics are exact. For example, if you do:

```
(factorial 1000)
```

It will actually display the full answer.