

Home
Essays
H&P
Books
YC
Arc
Bel
Lisp
Spam
Responses
FAQs
RAQs
Quotes
RSS
Bio
Twitter

PAUL GRAHAM

WHAT MADE LISP DIFFERENT

December 2001 (rev. May 2002)

(This article came about in response to some questions on the [LL1](#) mailing list. It is now incorporated in [Revenge of the Nerds](#).)

When McCarthy designed Lisp in the late 1950s, it was a radical departure from existing languages, the most important of which was [Fortran](#).

Lisp embodied nine new ideas:

1. Conditionals. A conditional is an if-then-else construct. We take these for granted now. They were [invented](#) by McCarthy in the course of developing Lisp. (Fortran at that time only had a conditional goto, closely based on the branch instruction in the underlying hardware.) McCarthy, who was on the Algol committee, got conditionals into Algol, whence they spread to most other languages.

2. A function type. In Lisp, functions are first class objects--they're a data type just like integers, strings, etc, and have a literal representation, can be stored in variables, can be passed as arguments, and so on.

3. Recursion. Recursion existed as a mathematical concept before Lisp of course, but Lisp was the first programming language to support it. (It's arguably implicit in making functions first class objects.)

4. A new concept of variables. In Lisp, all variables are effectively pointers. Values are what have types, not variables, and assigning or binding variables means copying pointers, not what they point to.

5. Garbage-collection.

6. Programs composed of expressions. Lisp programs are trees of expressions, each of which returns a value. (In some Lisps expressions can return multiple values.) This is in contrast to Fortran and most succeeding languages, which distinguish between expressions and statements.

It was natural to have this distinction in Fortran because (not surprisingly in a language where the input format was punched cards) the language was line-oriented. You could not nest statements. And so while you needed expressions for math to work, there was no point in making anything else return a value, because there could not be anything waiting for it.

This limitation went away with the arrival of block-structured languages, but by then it was too late. The distinction between expressions and statements was entrenched. It spread from Fortran into Algol and thence to both their descendants.

When a language is made entirely of expressions, you can compose expressions however you want. You can say either (using Arc syntax)

```
(if foo (= x 1) (= x 2))
```

or

```
(= x (if foo 1 2))
```

7. A symbol type. Symbols differ from strings in that you can test equality by comparing a pointer.

8. A notation for code using trees of symbols.

9. The whole language always available. There is no real distinction between read-time, compile-time, and runtime. You can compile or run code while reading, read or run code while compiling, and read or compile code at runtime.

Running code at read-time lets users reprogram Lisp's syntax; running code at compile-time is the basis of macros; compiling at runtime is the basis of Lisp's use as an extension language in programs like Emacs; and reading at runtime enables programs to communicate using s-expressions, an idea recently reinvented as XML.

When Lisp was first invented, all these ideas were far removed from ordinary programming practice, which was dictated largely by the hardware available in the late 1950s.

Over time, the default language, embodied in a succession of popular languages, has gradually evolved toward Lisp. 1-5 are now widespread. 6 is starting to appear in the mainstream. Python has a form of 7, though there doesn't seem to be any syntax for it. 8, which (with 9) is what makes Lisp macros possible, is so far still unique to Lisp, perhaps because (a) it requires those parens, or something just as bad, and (b) if you add that final increment of power, you can no longer claim to have invented a new language, but only to have designed a new dialect of Lisp ; -)

Though useful to present-day programmers, it's strange to describe Lisp in terms of its variation from the random expedients other languages adopted. That was not, probably, how McCarthy thought of it. Lisp wasn't designed to fix the mistakes in Fortran; it came about more as the byproduct of an attempt to axiomatize computation.

■ [Japanese Translation](#)
