A Short History of PL's (MacLennan 1999)

“Zeroth Generation” (1940's-50's): machine language / assembly: if/then, read/write, store, goto

```
1107      46     1112      1061
GETA      DATA   I         TMP
```

First Generation (1957): FORTRAN: named variables, “natural” syntax, (explicit) loops and conditionals, efficient compilation

```
      DIMENSION DTA(900)
      SUM = 0.0
      READ 10,N
10    FORMAT(I3)
      DO 20 I=1,N
      READ 30,DTA(I)
30    FORMAT(F10.6)
      IF (DTA(I)) 25,20,20
25    DTA(I) = -DTA(I)
20    CONTINUE
      DO 40 I = 1,N
      SUM = SUM + DTA(I)
     CONTINUE
      AVG = SUM/FLOAT(N)
      PRINT 50,AVG
      FORMAT(1H, F10.6)
      STOP
```

Second Generation (1960): ALGOL: type declarations, hierarchical structure, procedures

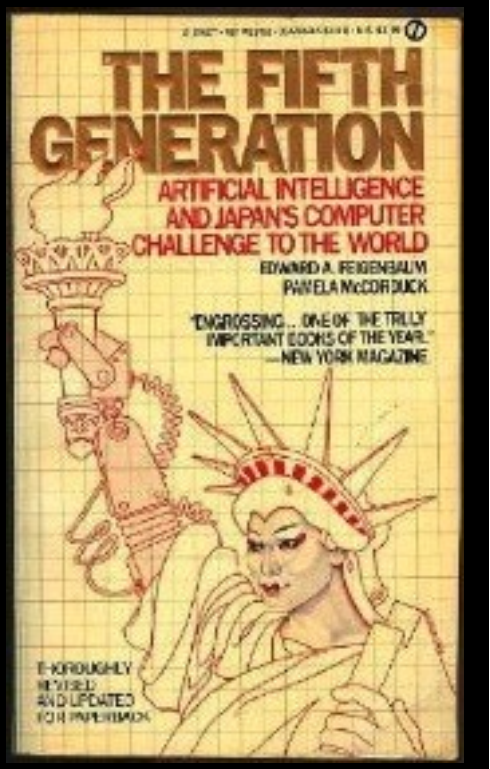
```
begin
  integer N
  ReadInt(N);

  begin
    real array Data[1:N];
    real sum, avg;
    integer i;
    sum := 0;

    for i := 1 step 1 until N do
      begin real val;
        ReadReal(val);
        Data[i] := if val<0 then -val else val
      end;
```

Third Generation (1970): Pascal: “return to simplicity” for teaching; efficiency, reliability

Fourth Generation (1979): Ada: object-oriented, secure (DOD), modular

Fifth Generation (1980s-present): programming *paradigms*

- Functional (LISP/Scheme, ML, Haskell)
- Object-oriented (Smalltalk, C++, Java)
- Logical (Prolog)



★★★☆☆ **Great Historical Document**, May 17, 2009

By **Better Yeti** (Hollywood, CA) - See all my reviews

This review is from: The Fifth Generation (Paperback)

This book is a really great specimen of the tech-think of its time. Feigenbaum had a hyped-up vision and a line of rhetoric that was like crack cocaine to corporate strategists of the mid-80s. It was also nicely complementary to the Japan-is-going-to-bury-us hysteria of that time. That Feigenbaum's vision was a pipe-dream, and his execution a train-wreck, are now history and public record. But I was there at the time; I was a Knowledge Engineer for one of his companies. I can tell you that Fortune-100 companies were just *throwing* money at these KE projects like there was no tomorrow. It was breathtaking. If nothing else, this book should be a great example of a red-hot business fashion and corporate hysteria that came, and went, with ultimately little to show for itself. Sound and fury, signifying nothing. And while I won't do a full Shakespeare and label this a tale told by an idiot, I think it's fair to say that Ed Feigenbaum had some truly epic blind spots and narcissism that you'll seldom find outside of politics or academia.

Lambda Calculus: The Essence of PL

- Variables: *x*, *y*, *z*
- Constants: 1, 2, 3
- Functions: +, *
- *Abstraction*: λ *x*. (* *x* *x*)
- *Application*: ((λ *x*. (* *x* *x*)) 3) = 9

(+ 3 4) = 7

Scheme: A Practical λ Calculus

- Use **`lambda`** instead of **`λ:`**

```
lambda(x)(* x x)
```

- Create named variables, functions w/ **`define:`**

```
(define square (lambda(x)(* x x)))
(define pi 3.14159)
```

- Support control via **`if:`**

```
(define negative?
    (lambda (x)
      (if (< x 0) ; if
          1       ; then
          0)))    ; else
```

Scheme: Lists + Recursion

- *List* (intuitively): An ordered set of objects:

```
'(abc 3.14159 17)
 (list 'abc 3.14159 17)
```

- *List* (formally):

  - '() is a list.

  - If *l* is a list and *a* is any object, then **(cons** *a l***)** is a list: *recursive definition*

- Anything that is not a list (or function) is an *atom*.

Scheme: `car`, `cdr`, `cons`

- **`car`**: the head (first element) of the list

```
(car '(1 2 3)) = 1
```

- **`cdr`**: the tail (rest) of the list

```
(cdr '(1 2 3)) = '(2 3)
```

- **`cons`**: list builder

```
(cons 1 '(2 3)) = '(1 2 3)

= (list 1 2 3)

= (cons 1 (cons 2 (cons 3 '())))
```