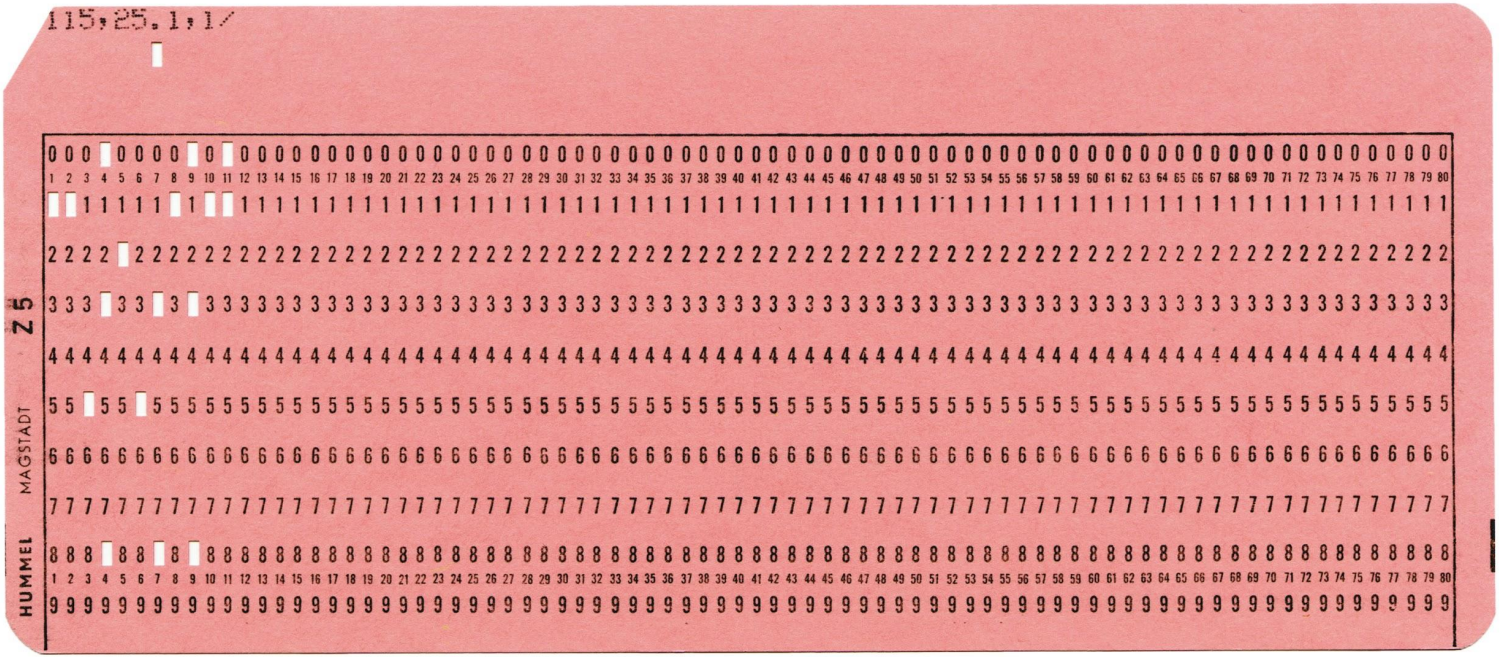
A little bit of LISP history

Nikolas Vourlakis

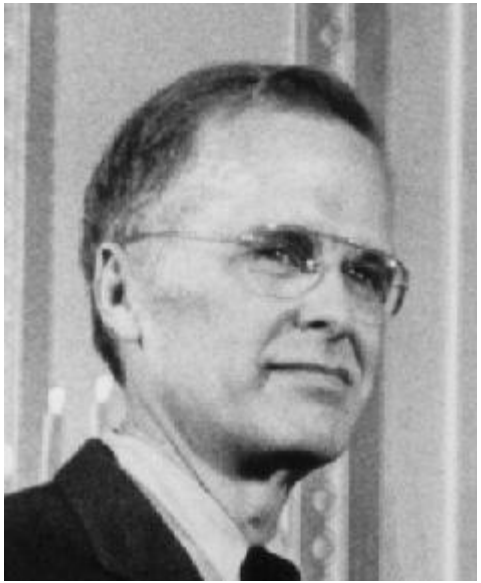
Lambda Calculus

- 1930: Introduced by Alonzo Church
  - Can simulate any turing machine
- 1936: Untyped lambda calculus
- 1940: Simply typed lambda calculus
- 1960: Relation to programming languages

The 50s



The 50s

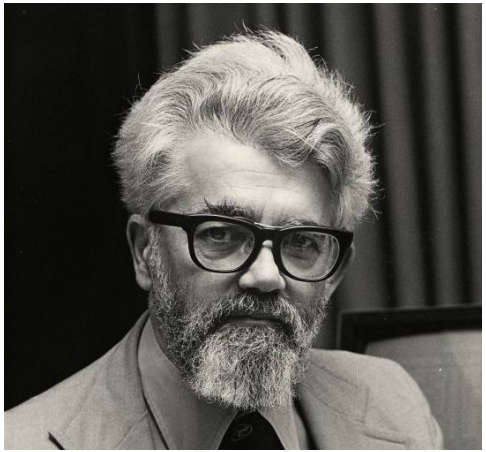


John Warner Backus

FORTRAN

- 1953: Proposed as an alternative to assembly
- 1954: The IBM Mathematical Formula Translating System
- 1957: First Compiler
- Initially treated with skepticism but quickly gained acceptance.

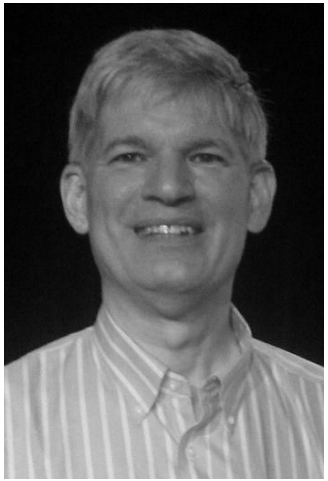
Mid 50s - 60s, LISP



John McCarthy

- Project Advice Taker
- 1958: Implementation of LISP begun
- 1960: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I
  - eval[e, a] function => interpreter
- Version 1.5
  - First compiler written in the language it compiled
- Adoption
- Dialects all over the place

The 70s, Scheme



Guy Lewis Steele Jr

Gerald Jay Sussman

- The “Lambda papers”
- Started as an experiment to understand the actor model
- Minimal LISP subset
- Correctly implemented lambda calculus
- Simulate all missing constructs with lambdas

80s - now

- LISP standardization
  - 1981: “LISP Community Meeting” by DARPA => Common Lisp
  - 1984: Published “Common LISP: The Language”
  - The first language standardization process done only through email!
- 1994: Racket
- 2007: Clojure

What made LISP different and unique

- Conditionals (1st)
- Function type
- Recursion (1st)
- A new concept of variables
- Garbage collection (1st)
- Programs are composed by expressions

- Code is data
- The whole language is always available

Language

- Atoms - indivisible data object
  - a
  - abc5
  - 12
- Lists - sequence of S-Expressions
  - ()
  - (FOO 43 BAR)
  - (* (+ 2 6)
     (- 7 5))

```
(display “Hello, World!”)
```

```
(define (fact n)
  (if (= n 1)
    1
    (* n (fact (- n 1)))))

(fact 100)
```

```
(map
  (lambda (x) (* 2 x))
  (list 1 2 3 4))
```

Extending the language - Macros

```
(define-syntax when
  (syntax-rules ()
    ((_ pred b1 ...)
     (cond [pred (begin b1 ...)]))))
```

```
(when (= i 10)
  (display "step 1")
  (newline)
  (display "step 2")
  (newline))
```

```
(cond
  [(= i 10)
   (begin
    (display "step 1")
    (newline)
    (display "step 2")
    (newline))])
```

Extending the language - Macros

```
1 (define-syntax my-if
2   (syntax-rules (then else)
3     [(my-if e1 then e2 else e3)
4      (if e1 e2 e3)]))
5
6 (my-if (= i 0)
7        then (display "then")
8        else (display "else"))
```