CS339: Abstractions and Paradigms for Programming

Bindings and Environment

Manas Thakur
CSE, IIT Bombay

Autumn 2025

QUIZ
TIME !

Programming Languages

- Syntax
  - Defined by a (context-free) grammar
- Semantics
  - Meaning of each construct that is enabled by the grammar

Design

- Evaluator/Interpreter
  - Execute the programs written using the syntax based on the defined semantics

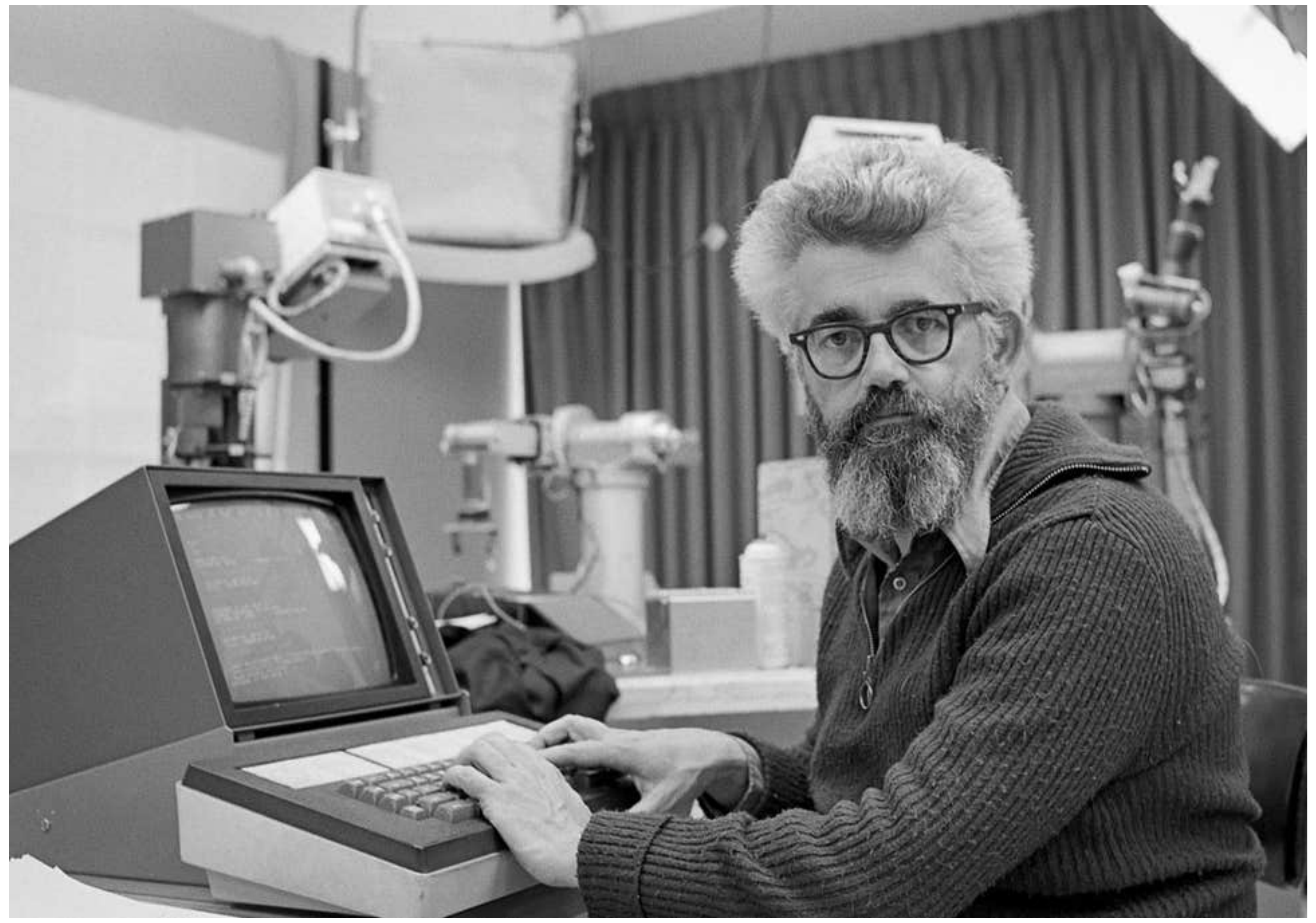
Implementation

Our primary vehicle for APP: **Scheme**

- A dialect of LISP (for LISt Processing)
  - The second oldest PL that is still in use today
  - Which is the first one?
- Our evaluator: DrRacket
  - A popular cross-platform tool to learn and design Lisp-like languages
  - Extremely popular in academia

Recalling another giant

- Coined the term *Artificial Intelligence*
- Designed several PLs
  - Invented LISP
  - Influenced the design of ALGOL
- Invented *garbage collection*
- Turing Award in 1971

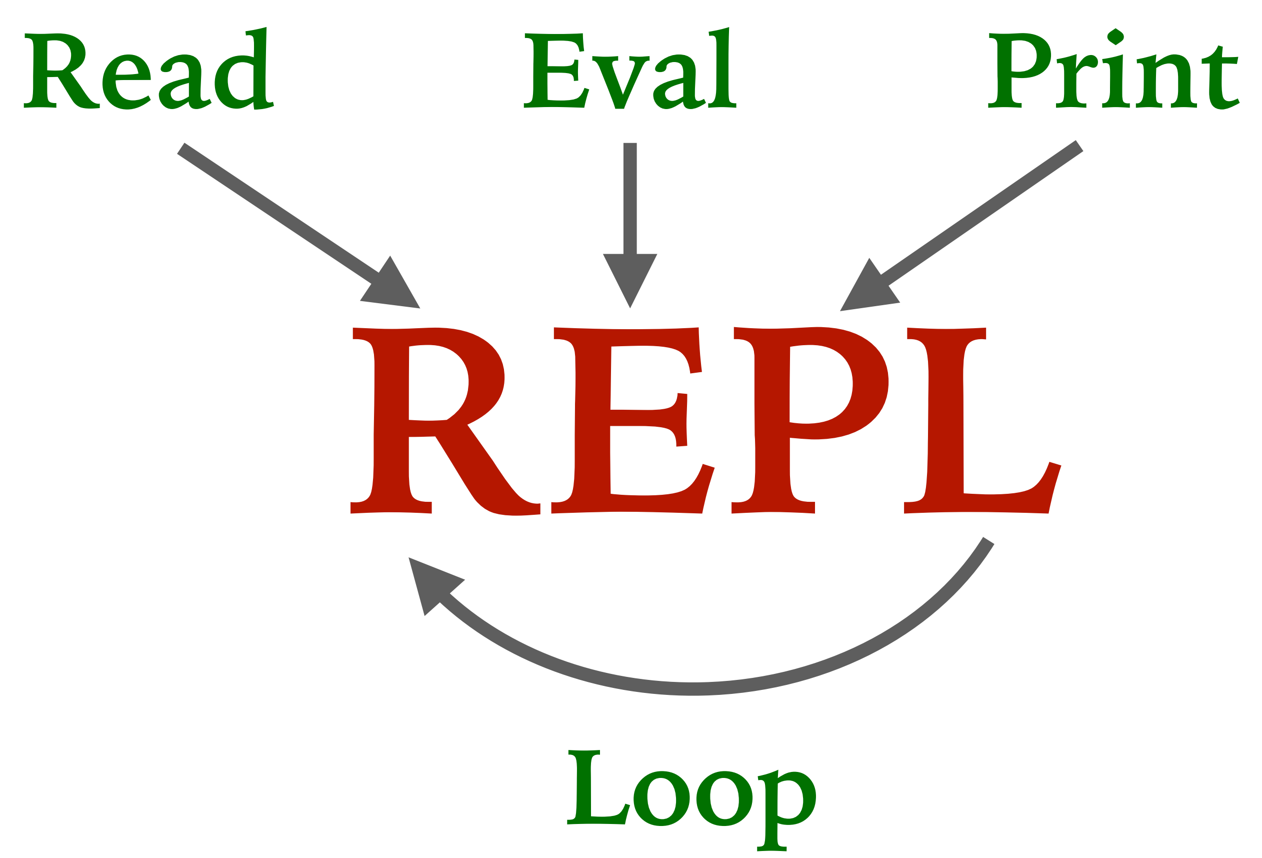

John McCarthy (1927-2011)

The Elements of a PL

- Primitive expressions
    - Things whose semantics are pre-defined in the language
- Means of combination
    - Building compound elements from primitive ones
- Means of abstraction
    - Ability to use compound elements as primitives themselves

Primitive expressions in Scheme

- An expression that cannot be evaluated further
- The language implementation (evaluator or interpreter) is "born" with these expressions
- How do interpreters work?



Combining Expressions

- Primitive **procedures** can be used to combine primitive expressions
- Result is a compound expression
- Syntax: `(operator operand1 operand2 ...)`

  Prefix notation
- Advantages of prefix notation?

Defining Abstractions

- Associate **names** with values

```
> (define x 2)
```

Evaluate:

```
> x
```

- Associate names with expressions

```
> (define y (+ x 1))
```

Evaluate:

```
> y
```

- Associate names with procedures

```
> (define (add x y)
    (+ x y))
```

Evaluate:

```
> (add op1 op2)
```

Procedures as Entities

- What did we associate the name add with here?

```
> (define (add x y)
    (+ x y))
```

- Can be equivalently written as: lambda is a keyword used to create anonymous functions i.e. functions without any name

```
> (define add
    (lambda (x y)
      (+ x y)))
```

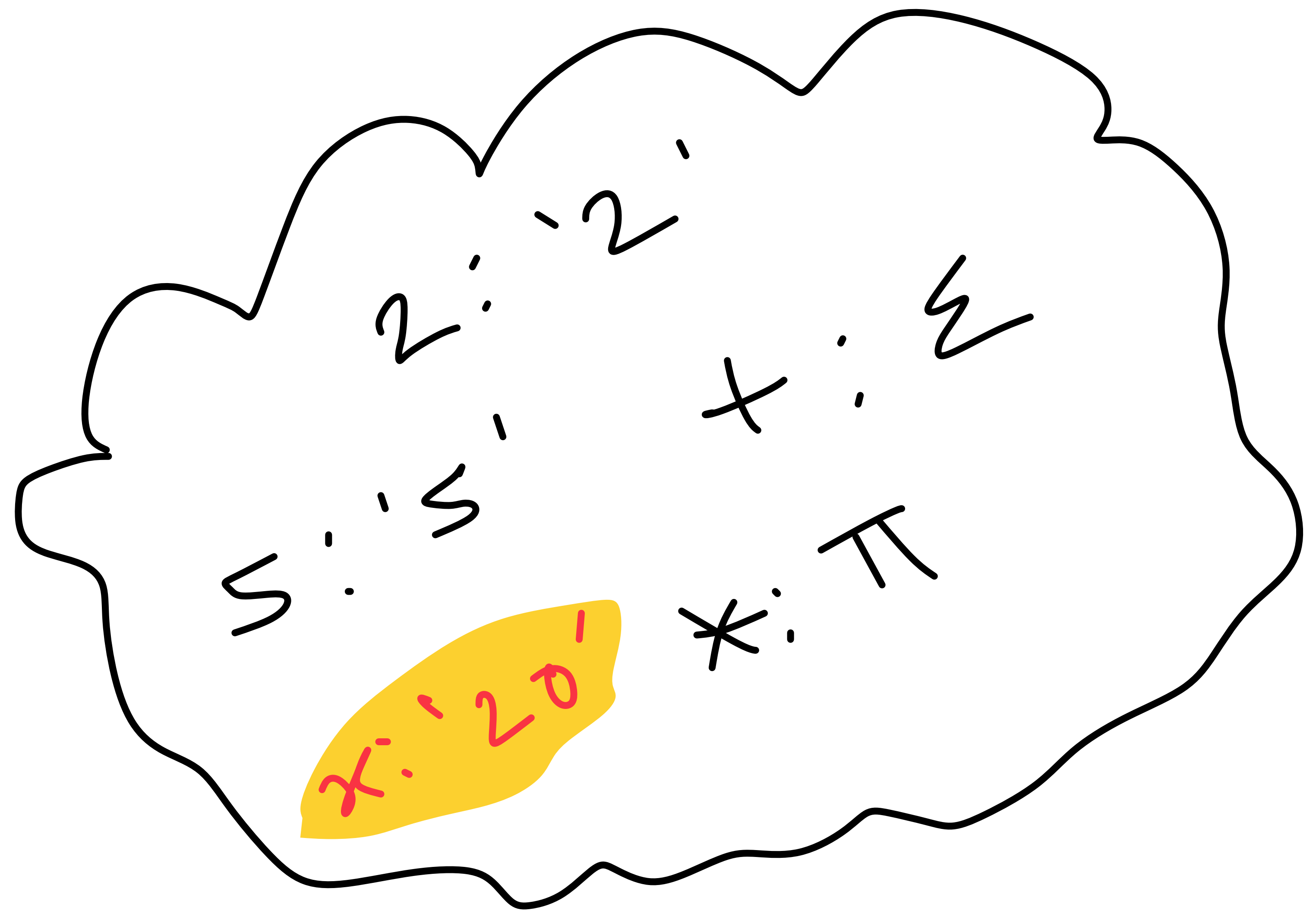
add belongs as much to the vocabulary now as + did!

- We had named a procedure (a lambda).
- Procedures/functions/lambdas are *first-class* entities even without names, in the exact same way as numbers — they *exist*!

Bindings and Environment

Binding = naming something so you can use it later.

- `(define x 20)` creates a binding from `x` to `20`
- Where is it created/remembered?
  - The **ENVIRONMENT**
  - Default: The **GLOBAL ENVIRONMENT**



- Lookup involves searching in the current environment
  - We shall see how do we work with multiple environments in future.

Applying Procedures

➤ Substitute actual arguments in place of formal parameters (at once) in the body of the procedure, and then evaluate the body of the procedure.

```
> (define (add x y)
    (+ x y))
> (add 4 (+ 7 5))
```

```
(+ 4 (+ 7 5))
(+ 4 12)
16
```

Next class:
Orders of Evaluation

Called the Substitution Model of Evaluation