Are there any non-recursive natural languages? What would happen to a society that spoke one?

Not for humans at least. They would run out of original things to say.

Chimps and Dolphins are able to learn non-recursive “languages” (some linguists argue they are not really “languages”), but only humans can learn recursive languages.

Language Elements

When learning a foreign language, which elements are hardest to learn?

- Primitives: lots of them, and hard to learn real meaning
- Means of Combination
  - Complex, but, all natural languages have similar ones (Chomsky)
  - SOV (45% of all languages): Sentence ::= Subject Object Verb (Korean)
  - SVO (42%): Sentence ::= Subject Verb Object
  - VSO (9%): Sentence ::= Verb Subject Object (Welsh)
  - OSV (<1%): Lladdodd y ddraig y dyn. (Killed the dragon the man.)
  - Tok Pisin (Papua New Guinea): mi (I), mitupela (he/she and I), mitripela (both of them and I), yumpela (you and I), yumpipela (all of you and I)
- Means of Abstraction: few of these, but tricky to learn differences across languages
  - English: I, we

Running out of Ideas

“It’s all been said before.”

Eventually true for a non-recursive language.

Never true for a recursive language. There is always something original left to say!
Why don’t we just program computers using English?

• Too hard and complex
  Non-native English speakers don’t need convincing. The rest of you have spent your whole life learning English (and first 5 years of your life doing little else) and still don’t know useful words like floccipocinniliphilification! There are thoughts that even native speakers find it hard to express.

By the end of today you will know enough Scheme (nearly the entire language) to express and understand every computation. By PS7, you will know enough to completely and precisely describe Scheme in terms of itself (try doing that in English!)

Why don’t we just program computers using English?

• Not concise enough

  English:
  To find the maximum of two numbers, compare them. If the first number is greater than the second number, the maximum is the first number. Otherwise, the maximum is the second number.

  Scheme:
  (define (max a b) (if (> a b) a b))

Why don’t we just program computers using English?

• Limited means of abstraction

  There are only a few pronouns: he, she, it, they, these, … (English doesn’t even have a gender-neutral pronoun for a person!) Only Webster and Oxford can make up new ones.

  define allows any programmer to make up as many pronouns as she wants, and use them to represent anything.

Essential Scheme

Expression ::= (Expression₁ Expression*)
Expression ::= (if Expression₁ Expression₂ Expression₃)
Expression ::= (define name Expression)
Expression ::= Primitive
Primitive ::= number
Primitive ::= + | - | * | ...
Primitive ::= ...

The exact meaning(s) of every Scheme expression is determined by simple, unambiguous rules we will learn today (and refine later in the course). Grammar is clear, just follow the replacement rules. But what does it all mean?
Expressions and Values

- (Almost) every expression has a value
  - Have you seen any expressions that don't have values?
- When an expression with a value is evaluated, its value is produced

Evaluation Rule 1: Primitives

If the expression is a primitive, it is self-evaluating.

> 2
> #t
> +
> <primitive:+>

Evaluation Rule 2: Names

If the expression is a name, it evaluates to the value associated with that name.

> (define two 2)
> two
> 2

Evaluation Rule 3: Application

3. If the expression is an application:
   a) Evaluate all the subexpressions of the combination (in any order)
   b) Apply the value of the first subexpression to the values of all the other subexpressions.

   (expression\textsubscript{0}, expression\textsubscript{1}, expression\textsubscript{2} ... )

Rules for Application

1. If the procedure to apply is a primitive, just do it.

2. If the procedure is a compound procedure, evaluate the body of the procedure with each formal parameter replaced by the corresponding actual argument expression value.
Making Procedures

- **lambda** means “make a procedure”

Expression ::= 
(lambda (Parameters) Expression)  
Parameters ::= 
Parameters ::= Name Parameters

Lambda Example: Tautology Function

(lambda () #t) with no parameters
(lambda () #t) with body #t
> ((lambda () #t) 150) 
#<procedure>: expects no arguments, given 1: 150
> ((lambda () #t))
#t
> ((lambda (x) x) 150)
150

You’ve Already Used Lambda!

(define (closer-color? sample color1 color2)
  Expr)

is a shortcut for:

(define closer-color? (lambda (sample color1 color2) Expr))

Evaluating Special Forms

**Eval 4-if.** If the expression is
(if Expression1 Expression1 Expression2)
evaluate Expression1. If it evaluates to #f, the value
of the if expression is the value of Expression2. Otherwise, the value of the if
expression is the value of Expression2.

**Eval 4-lambda.** Lambda expressions self-
evaluate. (Do not do anything until it is
applied.)
More Special Forms

- **Eval 4-define.** If the expression is 
  \((\text{define } \text{Name } \text{Expression})\)
  associate the Expression with Name.
- **Eval 4-begin.** If the expression is 
  \((\text{begin } \text{Expression}_0 \text{Expression}_1 \ldots \text{Expression}_k)\)
  evaluate all the sub-expressions. The value of the begin expression is the value of \(\text{Expression}_k\).