The Logo Programming Language

A teaching language: designed for introductory programming

One syntactic form for all purposes: invoking a procedure

Only two data types: words and sentences

Code is data: a line of code is a sentence

An elegant tagline: no threshold, no ceiling

A bit of fun: turtle graphics

Demo
Logo is a Dialect of Lisp

What are people saying about Lisp?

• "The greatest single programming language ever designed."
  - Alan Kay (from the UI video), co-inventor of Smalltalk

• "The only computer language that is beautiful."
  - Neal Stephenson, John's favorite sci-fi author

• "God's programming language."
  - Brian Harvey, Father of CS 61A

http://imgs.xkcd.com/comics/lisp_cycles.png
Call expressions are delimited by spaces

Logo procedures are equivalent to Python functions
• A procedure takes inputs (arguments) that are values
• A procedure returns an output (return value)
• A procedure may output None to indicate no return value

```
? print 5
5

Multiple expressions can appear in a single line

? print 1 print 2
1
2```
Nested Call Expressions

The syntactic structure of expressions is determined by the number of arguments required by named procedures.

- **print** takes one argument (input)
- **sum** takes two inputs
- **difference** takes two inputs too

\[
\begin{align*}
? \text{print} & \quad \text{sum} 10 \quad \text{difference} 7 & 3 \\
& 14
\end{align*}
\]

One nested call expression

**versus**

Two expressions on one line

\[
\begin{align*}
? \text{print} 1 & \quad \text{print} 2 \\
& 1 \\
& 2
\end{align*}
\]
Words are strings without spaces, representing text, numbers, and boolean values

```plaintext
? print "hello
hello
? print "sum
sum
? print "2
2
```

Sentences are immutable sequences of words and sentences

```plaintext
? print [hello world]
hello world
? show [hello world]
[hello world]
```
Sentence (List) Processing in Logo

Sentences can be constructed from words or sentences

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence</td>
<td>Output a sentence containing all elements of two sentences. Input words are converted to sentences.</td>
</tr>
<tr>
<td>list</td>
<td>Output a sentence containing the two inputs.</td>
</tr>
<tr>
<td>fput</td>
<td>Output a sentence containing the first input and all elements in the second input.</td>
</tr>
</tbody>
</table>

Demo
Expressions are Sentences

The run procedure evaluates a sentence as a line of Logo code and outputs its value

? run [print sum 1 2]
3

Its argument can be constructed from other procedure calls

? run sentence "print [sum 1 2]
3

? print run sentence "sum sentence 10 (run [difference 7 3])
14
Procedures

Procedure definition is a special form, not a call expression

```logo
? to double :x
> output sum :x :x
> end

? print double 4
8
```

Procedures are not first-class objects in Logo; they can only ever be referenced by their original procedure name

Procedure names can be inputs or outputs
Conditional Procedures

If and ifelse are regular procedures in Logo

*Meaning:* They do not have a special evaluation procedure

They take sentences as inputs and run them conditionally

```logo
? to reciprocal :x
> if not :x = 0 [output 1 / :x]
> output "infinity
> end

? print reciprocal 2
0.5

? print reciprocal 0
infinity
```
Dynamic Scope

When one function calls another, the names bound in the local frame for the first are accessible to the body of the second.

No isolation of formal parameters to function bodies, as we saw with lexical scope:

? to print_x :x
> print_last_x
> end

? to print_last_x
> print :x
> end

? print_x 5
5
Logo Examples

Demo
Homework: Huffman Encoding Trees

Efficient encoding of strings as ones and zeros (bits).

```
A 0     C 1010   E 1100   G 1110
B 100   D 1011   F 1101   H 1111
```

Decoding a sequence of bits:

```
1 0 0 0 1 0 1 0
```

B  A  C