

## Experimenting with Language in Lynx

Run Lynx in your browser at <https://lynxcoding.club>

Lynx is a dialect of the Logo programming environment with multiple turtles, a large primitive vocabulary, and a browser-based interface.

---

Logo procedures come in two species, commands and reporters.

**Commands** do something. You have worked mostly with commands up until this point. `forward`, `right`, and `repeat` are all commands.

**Reporters** report something or return some result. Arithmetic operations, `heading`, and `color` are reporters.

All Logo expressions must begin with a command, otherwise you will generate an error message. For example, `forward 100 + 50` is fine. `100 + 50` will cause Logo to say, *I don't know what to do with 150.*

Procedures built into Logo are called **primitives**. You can also define your own procedures.

Some procedures/primitives have inputs while others do not. This is true for both commands and reporters

- `Clean` is a command with zero inputs
- `Forward` is a command with one input.
- How many inputs does the command `repeat` have?

**Words** in Lynx begin with quotation marks as in:

```
show "Gary
```

Lynx **lists** are a collection of words or other lists, such as:

```
show [lemon grape [apple pie] strawberry]
```

The list above has 4 elements, 3 words and 1 list. A good deal of computer programming involves taking things apart and putting things together. In this activity, we will take things apart.

1) Try the following in the command center:

```
show first "apple  
show last "apple
```

What does the reporter, `first`, do?

What does the reporter, `last`, do?

2) Predict what each of these instructions will do before you try them.

```
show first [apple peach pear]  
show last [apple peach pear]  
How accurate were your predictions?
```

3) What do you think will happen if you type the following? Make a prediction and then run the instructions in the command center. Write the results next to the instruction.

```
show first first [apple peach pear]  
show last first [apple peach pear]  
show first last [apple peach pear]  
show last last [apple peach pear]
```

How accurate were your predictions?

4) Predict the result of the following instructions before typing them into the command center. Write the results next to the instruction.

```
show bf "lemon  
show bl "grape  
show bf bf "grape  
show bl bf "grape  
show bf bl "grape  
show bl bl "grape
```

What does `bf` do? \_\_\_\_\_

What does `bl` do? \_\_\_\_\_

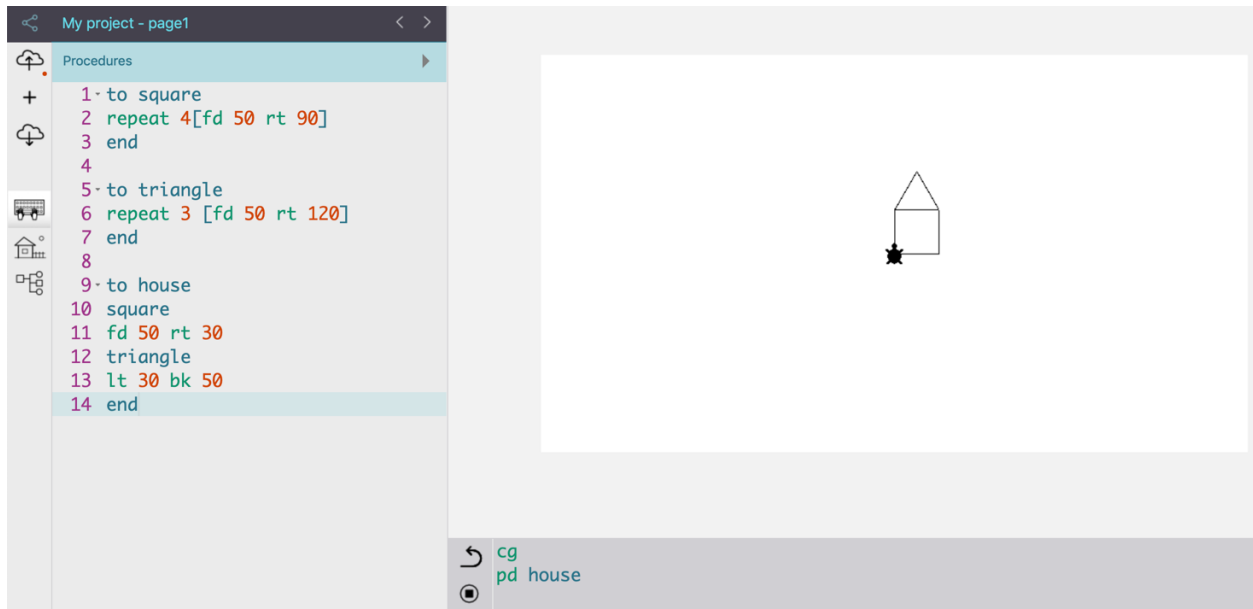
5) Predict the result of the following instructions before typing them into the command center. Write the results next to the instruction.

```
show bf [apple grape peach]
show bl [apple grape peach]
show bf bf [apple grape peach]
show bl bf [apple grape peach]
show bf bl [apple grape peach]
show bl bl [apple grape peach]
show bf bl [apple grape peach]
```

6) Predict the result of the following instructions before typing them into the command center. Write the results next to the instruction.

```
show first bf "grape
show first bl "grape
show last bf bf [apple grape peach]
show first bl bf [apple grape peach]
show first bf bl "grape
show first bl bl "grape
show last bf bl "grape
```

## Writing procedures in Lynx



Procedures are lists of instructions with a name.

- Those names need to be a single word. Multiple words may be smushed together without spaces.

They begin with the word, *to*, and end with the word, *end*.

*to* and *end* need to be on their own line in the procedures center. For example...

```
to square
repeat 3 [fd 50 rt 120]
end
```

Once you write a procedure in the procedures center, test it by typing its name in the command center. Did it work correctly? If not, debug the program.

You run a procedure by typing its name (and any necessary input values) in the command center.

Type the following in the command center:

```
show pick [cherry apple peach]
show pick "abc
```

What does `pick` do?

## Gossip

Create these procedures

```
to person
output pick [Gary Amanda Bill Sylvia Mary Joe]
end
```

```
to doeswhat
output pick [punches loves dislikes kicks hugs]
end
```

output is a reporter that takes an input and reports its result.

Try the following in the command center:

```
show person
show doeswhat
show person doeswhat
show se person doeswhat
show se person doeswhat person
show (se person doeswhat person)
```

What do each do? Are errors generated?

- What does `se` or `sentence` do?
- When do you need `( )` around sentences?

SE is short for sentence. It's a reporter that takes two or more inputs (with parentheses) as input and outputs them as a sentence.

Add this new procedure to your procedures.

```
to gossip
show (se person doeswhat person)
end
```

Try the following in the command center:

```
repeat 5[gossip]
```

### Challenge:

Can you add procedures to randomly generate adjectives and/or adverbs? Name them `adjective`, `adverb` or `how` and `kindof`. You may also change the name of the `person` and `doeswhat` procedures to `noun` and `verb` if you wish.

Where would you place those new reporters in the `gossip` procedure?

How would you change the procedures to make them insult one person more often than others, while maintaining randomness?

Add multiword actions by using lists within lists, such as the following.

```
to doeswhat
output pick [punches loves dislikes [looks at] kicks hugs [plays
ball with] [dances like]]
end
```

### **Literary Challenge**

Write procedures to generate random haiku or other forms of poetry!

```
to haiku
show (sentence 5syllables 7syllables 5syllables)
end
```

## Possessive Machine

You define reporter procedures with inputs by adding an input name, preceded by a :, in the title line of procedures. Here is an example.

```
to double :number
output :number + :number
end
```

or

```
to double :number
output :number * 2
end
```

Try the following instructions in the command center:

```
show double 5
show double double double 5
```

What does `double` do?

When a procedure hits an `output`, that procedure ends as the procedure tosses out some value, hoping a command catches it.

Lots of programming is comprised of taking stuff apart and smushing things together. The following challenges will do a bit of both.

```
to possessive :thing
output word :thing "'s
end
```

(*Note* I used quotation marks and an apostrophe next to each other)

Try the following in the command center:

```
show possessive "Gary
show possessive "boys
```

Do you see the bug in your `possessive` procedure?

Let's fix it!

```
to possessive :thing
if (last :thing) = "s [output word :thing "' ]
output word :thing "'s
end
```

What is happening in this new version of the `possessive` procedure?

*Note:* the input name *thing* is arbitrary. It could be `x` or `y` or `word`.

**Super colossal challenge!**

Can you write a plural procedure that takes a word as input and exports its plural form?

Be sure to check for lots of exceptions since English is tricky.