PROCEDURES, FUNCTIONS AND CONTROL STRUCTURES

Today

- Answer questions on:
  - Midterm
  - Homework for Unit B, Homework for Unit C
- Talk about variables, expressions and functions in the context of Netlogo.
- Remind you to do the readings for Units D, E and F.

Any questions?

Functions (first time)

- Functions are bits of program that generate *values*.
- Since they generate values, it is natural that we use them along with assignment.
- We use functions as a way to get *abstraction*.
- You can think of abstraction as "hiding the detail".
- Rather than writing out the Netlogo code for generating a random number every time that we want one, we just call `random`.
- `random` is provided by the folk who wrote Netlogo, but you can also write your own functions.

Procedures

- In fact we don't write many of our own functions in Netlogo.
- We do write *procedures*. Procedures are bits of code that *do something*:
- `catch-sheep` is a nice example.

```
to catch-sheep
  let prey one-of (sheep-here
                   with [not grabbed?])
  if prey != nobody
    [ set grabbed?-of prey true
      ask prey [ die ]
      set energy energy + wolf-gain-from-food
    ]
end
```

Procedures (more)

- A procedure starts with:

```
to name-of-procedure
```

  and ends with

```
end
```

- In between, the procedure contains a list of *instructions*.
- These instructions are the steps in the *algorithm* that the procedure uses.

Procedures (even more)

- You then *call* a procedure to make it execute.

```
to go
   :
  ask wolves [
     move
     set energy energy - 1
     catch-sheep
     reproduce-wolves
     death   ]
   :
end
```

- So one procedure is called by a second procedure which may be called by a third procedure, and so on ....

Procedures (last)

- Procedures can take inputs:

```
to color-sheep [this-many]
   repeat this-many
   [
      ask one-of sheep
         [set color red]
   ]
end
```

- To call this procedure, you have to give it a number (an integer) that sets the number of sheep to paint red:

```
color-sheep 10
```

Functions (again)

- In Netlogo, functions are called *reporters*.
- They report values.
- They are defined and called much like procedures:

```
to-report sheep-count
  report count sheep
end
```

- The difference between writing functions and procedures is that:
  - Functions start with `to-report`
  - Functions us `report` to *return* a value.

Functions (more)

- The value you get from a function is like any other value.
- You can use it in an expression:

```
set energy energy + sheep-count
```

- Or, slightly more sensibly:

```
if sheep-count > 300
   [stop]


if sheep-count > 300
   [ask sheep [die]]
```

Functions (even more)

- Just like procedures, you can write functions that take inputs.
- Let's imagine we want to limit the number of sheep that we have.

```
to cull-sheep [how-many-to-kill]
  repeat how-many-to-kill
    [
       ask one-of sheep
          [die]
    ]
 end
```

Functions (one last time)

- To use this procedure, we need to know how many sheep we want to kill
- A function can tell us this:

```
to-report cull-this-many [limit-on-sheep]
  report (count sheep) - limit-on-sheep
```

- We can then use the function and procedure together:

```
cull-sheep cull-this-many 200
```

Control structures (if)

- We use *control structures* in procedures to control what Netlogo does.
- For example:

```
to kill-red-sheep
   ask sheep
      [
         if (color = red)
            [die]
      ]
end
```

Control structures (if, again)

- In general, an `if` looks like:

```
if <something that is true or false>
    [
          :
      some instructions
          :
    ]
```

Control structures (if, more)

- The true/false bit can be more complicated
- For example:

```
to-kill-red-sheep
   ask sheep
     [
       if (color = red) and (sheep-count > 200)
          [die]
     ]
end
```

- What is this going to do?
- You can use `or` as well as of `and`

Control structures (if, finally)

- We can use `not` to change the condition of an `if` around:
- For example:

```
to-kill-red-sheep
   ask sheep
     [
       if (color = red) and not (sheep-count > 200)
          [die]
     ]
end
```

- What is this going to do?

Control structures (ifelse)

- We can add to the `if` with an alternative set of instructions if the true/false bit is false:

```
to-kill-red-sheep
  ask sheep
     [
       ifelse (color = red) and (sheep-count > 200)
          [die]
          [set color blue]
     ]
end
```

- How would you change this so that only the red sheep became blue?

Control structures (nested if)

- We can put one `if` "inside" another:

```
to-kill-red-sheep
  ask sheep
     [
       if (color = red)
          [
             ifelse (sheep-count > 200)
                [die]
                [set color blue]
          ]
     ]
end
```

- Now, if there are less than 200 sheep, the red ones will turn blue.

Control structures (repeat)

- Sometimes we want to have actions happen several times over.
- We use a `repeat` to do this.

```
to cull-sheep how-many-to-kill
  repeat how-many-to-kill
    [
       ask one-of sheep
          [die]
    ]
 end
```

- Look familiar?

Control structures (repeat, repeated)

- In general, an `repeat` looks like:

```
repeat <however many times you want>
    [
            :
      some instructions
            :
    ]
```

Summary

- This lecture talked about some of the computer science ideas behind Netlogo.
  - Procedures
  - Functions
  - Control structures