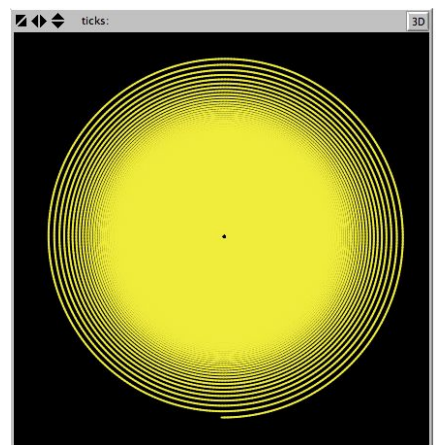
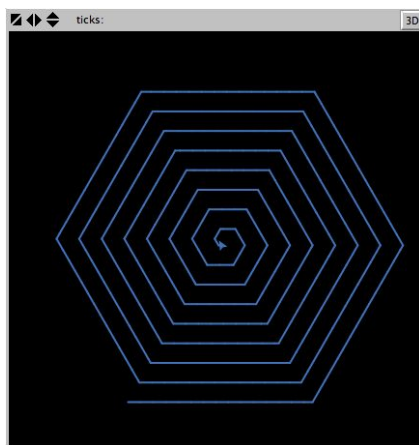
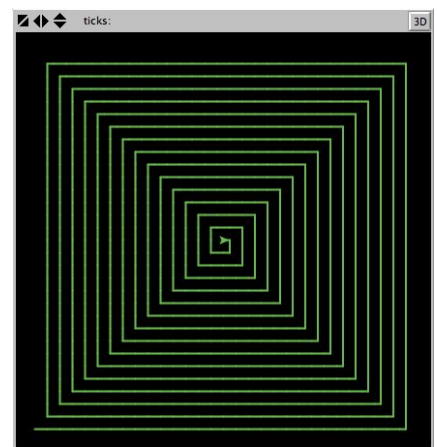
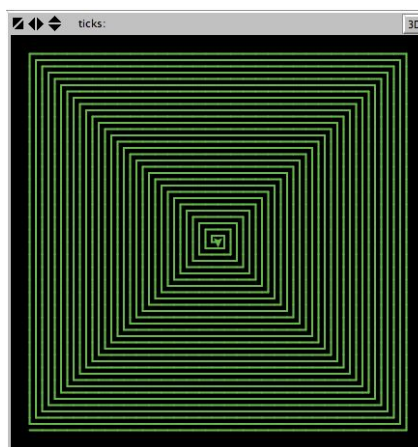
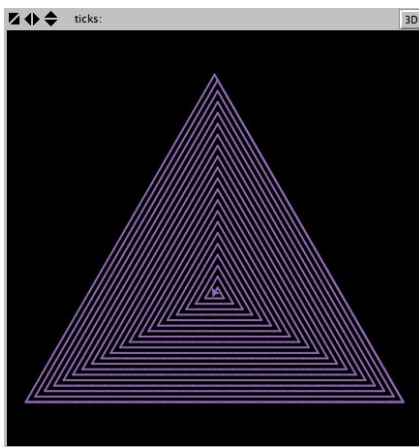


CS108L Computer Science for All

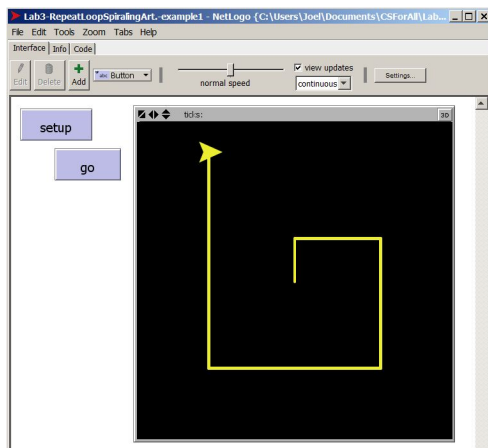
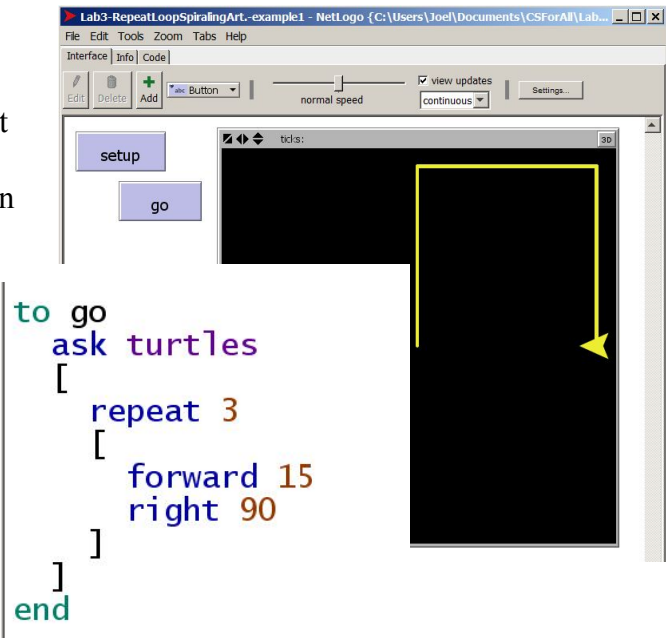
Module 2 Guide

Spiraling Geometry using Repeat Loops

In this lab, your assignment is to use NetLogo's *repeat loops*, *local variables* and *procedure arguments* to draw your own spiraling geometric shapes! Don't worry; you don't need to know any geometry to do this. All the information you need here in the lab.



In this lab you will be drawing geometric shapes (like triangles, squares, pentagons, etc.) using a NetLogo turtle in and at least one **repeat** loop. A repeat loop is a block of code that is repeated a set number of times. In the example to the right, the three yellow lines making part of a box are drawn using a repeat loop. The loop asks the turtle to move **forward 15** steps and turn **right 90** degrees and repeat that process three times (**repeat 3**). Notice that **square brackets** following the repeat command are used to tell the turtle which code is included in the repeat loop. Also notice that, unlike in Module 1, the “go” button is NOT set to go forever.



To make a spiraling geometric shape you will need to do at least three things inside the repeat loop: (1) draw a line, (2) change the heading of the turtle and (3) change the length of the line drawn.

The four-sided spiral shown to the left uses the same basic drawing commands as the image above: move forward and turn right by 90°. An **iteration** of a repeat loop means that the turtle completes the commands in the command block (within the square blocks) of the repeat loop one time. In each **iteration** of the loop below, the turtle moves forward, farther than it moved in the previous iteration. This can be done by replacing the “hardcoded” value of 15 in the

forward command with a **local variable** that gets larger with each iteration of the repeat loop. This requires using two new NetLogo commands:

let <i>variable (expression)</i>	Creates a new local variable AND sets its value to the value of the expression. The expression can be a single number or a calculation. For example: let x (0.5) let angle (360 / 4)
set <i>variable (expression)</i>	Changes the value of a local variable that <i>has already been created</i> to the value of the expression. For example: set x (1.5)

In the NetLogo code to the right:

- **let x (4.5)** creates a new local variable **x** and sets its value to 4.5
- **repeat 5** starts the repeat loop. The code within the square brackets after the repeat 5 statement is the code that is repeated 5 times.
- **forward x** moves the turtle forward a number of steps equal to the value of **x**. The first time through the repeat loop, the value of **x** is 4.5.
- **set x (x + 4.5)** changes the value of **x** to the result of the expression **(x + 4.5)**. The first time this command is reached, **x** has a value of 4.5 so the expression **(x + 4.5)** has a value of 9.0. Thus, the first time this command is reached, the value of **x** is changed from 4.5 to 9.0.
 - NOTE: In NetLogo, mathematical expressions (+, -, *, /) must have a space before and after the operators. For example, **(x+4.5)** is an error. **(x + 4.5)** is correct.
- **forward x** is inside the repeat loop and the second time it is reached, **x** has the value 9.0. Therefore the second line drawn by the turtle is 9.0 steps long.

```
to go
ask turtles
[
  let x (4.5)
  repeat 5
  [
    forward x
    right 90
    set x (x + 4.5)
  ]
end
```

In this lab you need to create at least three geometric spiraling shapes. Each spiral has to have a minimum of 20 lines. There is a simple rule to figure out the angle the turtle needs to turn to create the shape (the interior angle for the shape):

$$\text{Angle used} = 180 - \frac{360}{\text{number of sides}}$$

The values for some geometric shapes are provided in the table below.

Shape	Number of sides	Interior Angle
Triangle	3	60
Square	4	90
Pentagon	5	108
Hexagon	6	120

Procedure Arguments

Procedure arguments work very similar to local variables. Instead of declaring and assigning them with a **let** statement, the variables are declared within brackets immediately after the procedure's name. Any number of variables may be declared within the brackets (from 1 to as many as needed) , a space separates each variable. Below are some examples of setting up procedures with arguments. To use a procedure with arguments, call the procedure like normal and then give the values of all the arguments to be used in the procedure with spaces in between.

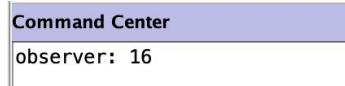
This is called ‘passing values to a procedure’. Note, the below examples use the command **show** statement. This is a handy way of checking what value is contained in the variable. It just ‘shows’ the value on the interface tab under the Command Center.

```
to sum3 [a_value b_value c_value]
  let sum_of_3 (a_value + b_value + c_value)
  show sum_of_3
end
```

The above procedure sum3, takes 3 arguments a_value, b_value and c_value. It then creates a new local variable called sum_of_3 and ‘shows’ the value in the command center. To call this procedure you type the name and then all the values you want to sum. Like below;

```
sum3 2 10 4
```

This would printout 16 in the command center (the procedure is an ‘observer’ procedure so it prints ‘observer’ first).



Similarly if you wanted to draw a line of different lengths with different turtles (already created). You could write a procedure that took the number of times to repeat a forward command and the turtle id like the drawLine procedure below.

```
to drawLine [repeatNumber turtleId]
  ask turtle turtleId [
    pen-down
    repeat repeatNumber [
      forward 1
    ]
  ]
end
```

When used with the following procedure call

```
drawline 5 0
```

This has turtle 0 put its pen-down (to draw) then moves forward 1 step 5 times which produces a line 5 steps long in the direction the turtle is pointing (right in this case).



As with other local variables you may modify the value contained in the argument with the use of the **set** command.

Here we multiply two numbers together and 'show' the result. However, prior to the multiplication we increase the value of `x_value` by 1.

```
to multiply [x_value y_value]
  set x_value (x_value + 1)
  let z_value (x_value * y_value)
  show z_value
end
```

When we call this procedure using;

multiply 3 5

We get the value of 20 not 15, since 3 is increased to 4 with the `set x_value (x_value + 1)` statement.

Command Center

observer: 20

Using arguments is very common in computer programming, in many cases a programmer will find themselves retyping code that varies only by a variable or two. In those cases, it is more proper to write a single procedure that takes as arguments the variables that are changing and just call this procedure when needed. A procedure built into NetLogo that is commonly used is **setxy** where the first argument is the x-coordinate of the turtle and the second argument is the y-coordinate of the turtle.