

The Absolute Beginner's Guide To

CODING

Using MLogo

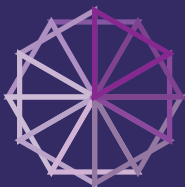


Table Of Contents

01	Why Learn To Code?	4 - 5
	All About Programming • Downloading MLogo	
02	Create Your First Program	6 - 10
	Learn about MLogo • Draw a Rectangle	
03	Deleteing Your Drawing	11-12
	clrturtle • home	
04	Comman Tasks	13
	A Table of Common Tasks Performed in MLogo	
05	Repeat Command	14-18
	Draw a Circle and a Triangle	
06	Nested Repeats	19-23
	Draw More Complicated Shapes	
07	Making Colors	24-27
	RGB values • random function	
08	All About Variables	28-32
	Addition • Subtraction • Multiplication • Division	
09	Awesome Functions	33 - 38
	Functions	

10

If Statements

39 - 48

if statments • conditional statements

11

Words and Printing

49 - 52

Create a Wacky Poem

12

Problem Solutions

54 - 60

Solutions to Practice Problems

Why Learn To Code?

01

All About MLogo

Have you ever wondered how websites are built, how video games are created, or how apps for smartphones are designed?

They were all made with a programming language, which is a special language that allows people to communicate with computers.

What is MLogo?

One of the many programming languages available is MLogo. People like to program with MLogo because it is easy to draw complex images.

When you write a program in MLogo, you are mak-

ing a list of instructions for the computer to follow. A computer's job is to follow instructions. In fact, a computer follows millions of instructions every second.

Your job is to learn how to communicate with the computer so it will follow your instructions. Don't worry, learning how to code is not hard at all, and you will soon be well on your way to creating your own computer programs.

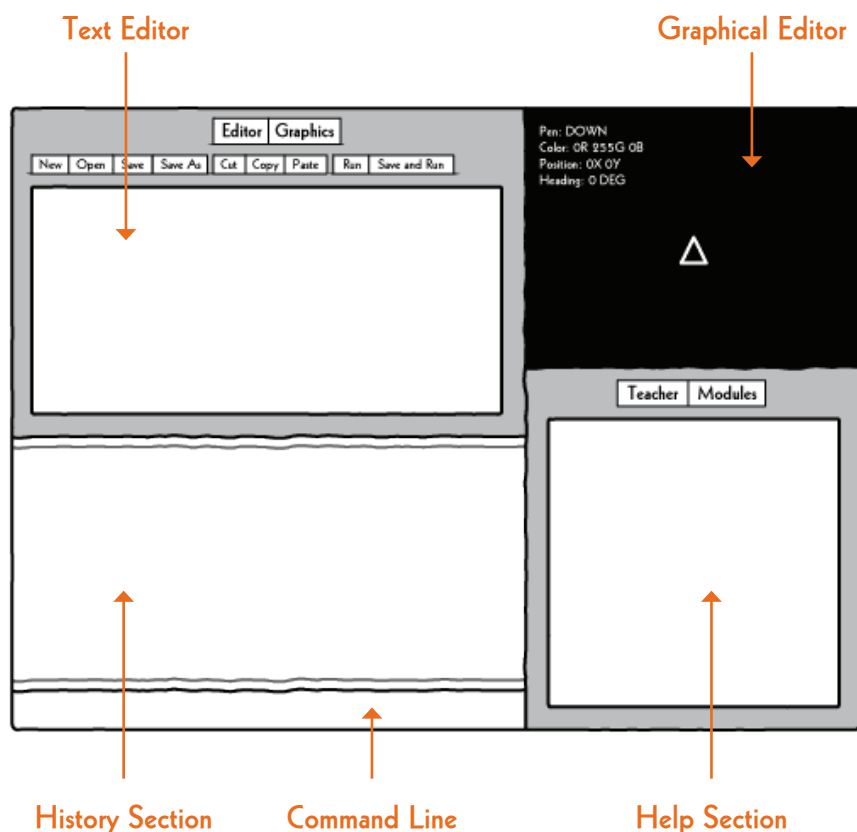
Download MLogo

MLogo is a free program that can be downloaded for your home computer from <https://sites.google.com/a/mtu.edu/ccp/curriculum/logo>. MLogo is available for Macs, Windows, and Linux.

Create Your First Program

02

When you first open MLogo you will see something similar to the sketch below.

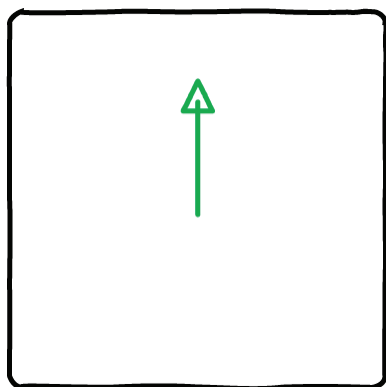
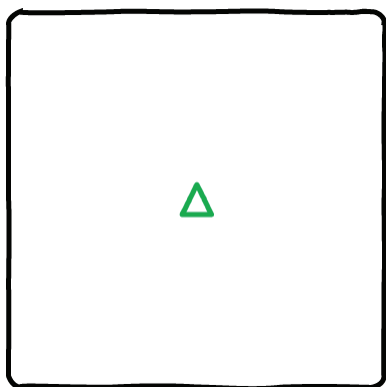


Step 1) Drawing in MLogo is a lot like drawing with a pen and paper, except in order to draw something in MLogo, you have to tell the computer in which direction to draw the line and how far to draw the line.

When you open the program, the first thing you will notice is that there is a little green triangle in the middle of the graphics screen. You can think of this triangle as a pen. Whichever direction the pen is pointing is the direction the line will be drawn. Our goal for our very first program is to draw a rectangle.

Step 2) The first thing we need to do is draw a straight line. In the text editor, type the following and then press the ENTER key on your keyboard.

```
forward 50
```

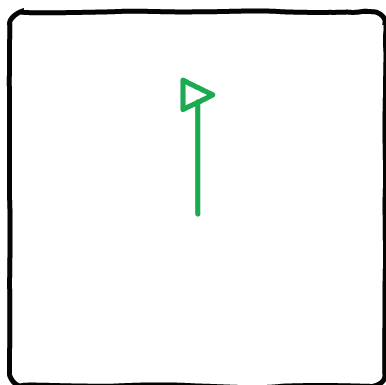
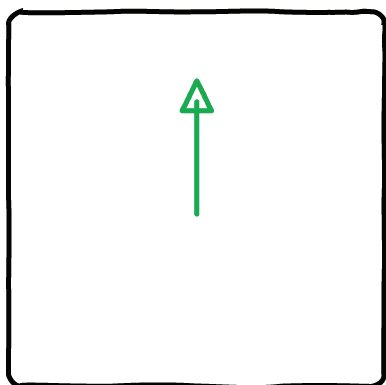


Step 3) The pen will move up the screen and there will be a line from where the pen started to where the pen ended.

Step 4) Now we want to draw the top of the rectangle. To change the direction of pen type the following:

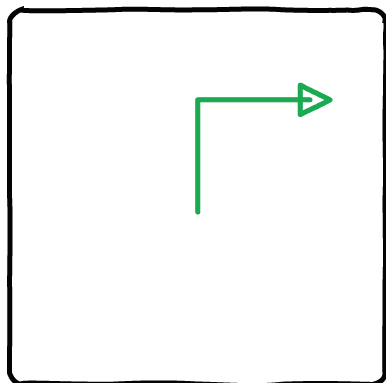
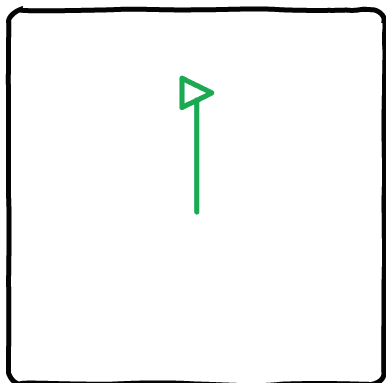
right 90

This tells the program to rotate the pen 90 degrees to the right. The pen will change direction and will now be facing right.



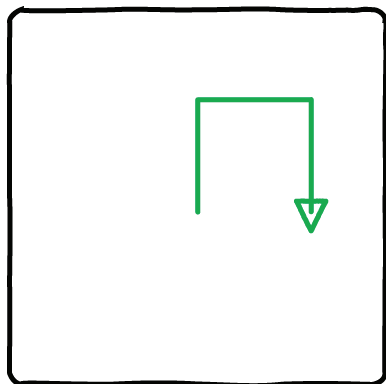
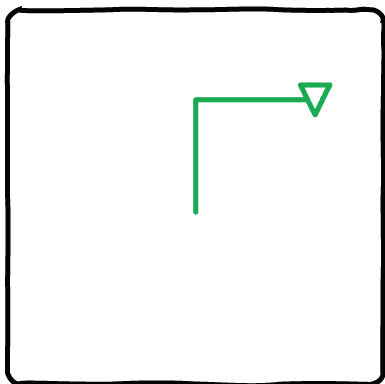
Step 5) Let's draw the top of the rectangle by telling the pen to go forward 50 pixels.

forward 50



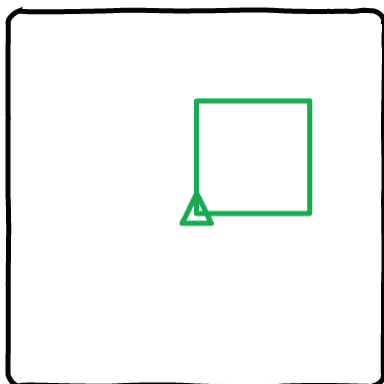
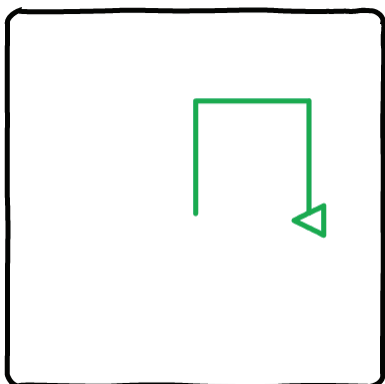
Step 6) Next, to draw the right side of the box:


```
right 90  
forward 50
```



Step 7) And finally, to draw the bottom of the box:

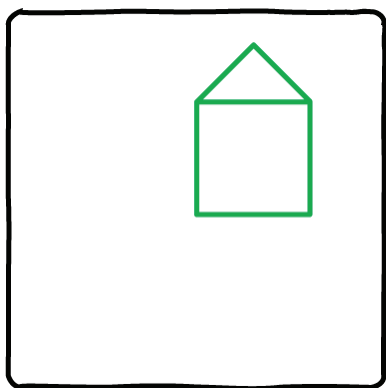
```
right 90  
forward 50  
right 90
```



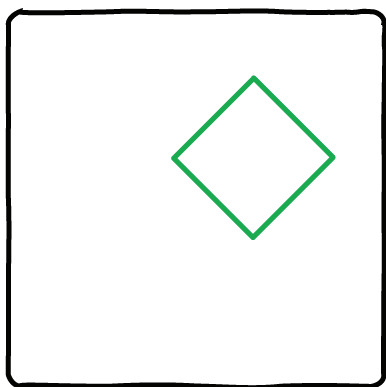
Step 8) We manage to create a rectangle! In future chapters we will learn how to draw triangles and circles.

Ch 2: Practice Problems

1) Can you draw a house by adding a triangle on top of the rectangle we drew in this chapter?



2) Try drawing a square that has been rotated 45 degrees. It is very similar to the rectangle that we drew in this chapter except the initial angle of the pen is 45 degrees instead of 90 degrees.

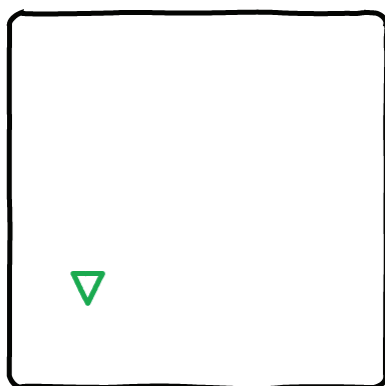
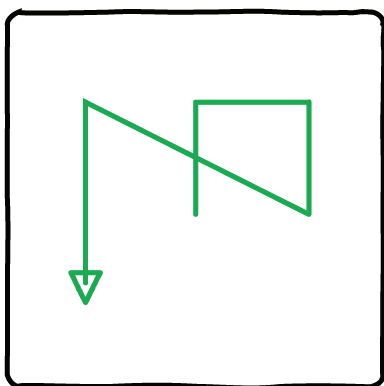


Deleting Your Drawing

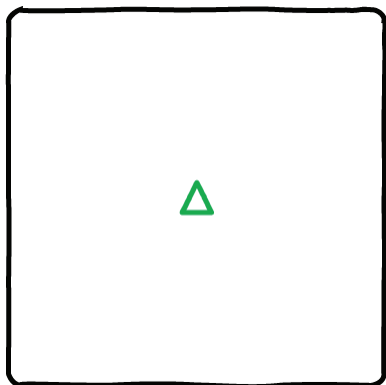
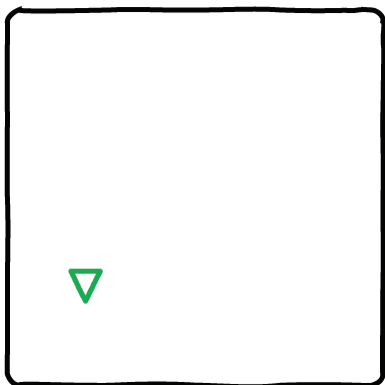
03

Clearing Your Drawing After you have experimented with drawing, you may want to clear your drawing board so you can start anew. The fastest way to clear the drawing board is to type:

```
clrturtle
```



Return Home If you want to return your triangle to the middle of the screen, you can type:



Hint: Type all commands with lower-case letters. For example, type 'forward 50', not 'FORWARD 50' or 'Forward 50.' MLogo can not tell that 'forward' and 'FORWARD' are the same word so MLogo will issue a warning like this in the Help Section: “**runtime error: I don't know how to 'FORWARD'!**”

Common Tasks

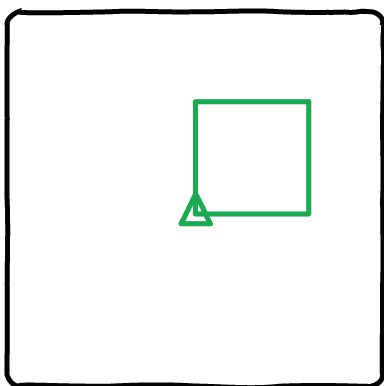
04

Common Commands

Command	The command does:
forward	Go forward
back	Go backward
right	Rotate pen to right
left	Rotate pen to left
clturtle	Clear the drawing
home	Return pen to the middle of screen
penup	Do not draw a line as pen moves
pendown	Draw a line as pen moves
help	Get a list of common commands

Repeat Command

05



You may have noticed that in the first program we typed the commands “forward 50” and “right 90” four times each in the first program. There is a way to shorten this task so we only have to type each command once. We can do this by using the **repeat** command.

Instead of using the text editor to type one command in at a time, we are going to type in the type box (see page 6 for a drawing of the location). Type the following:

```
clrturtle
```

```
home
```

```
repeat 4 [
```

```
    forward 100
```

```
    right 90
```

```
]
```

Clear the previous drawing

Return pen to middle of screen

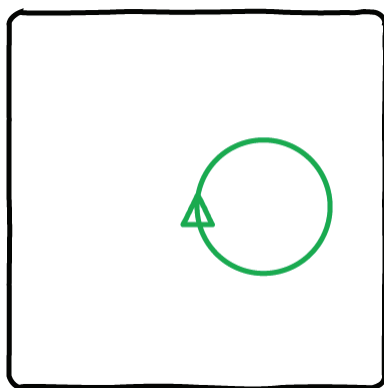
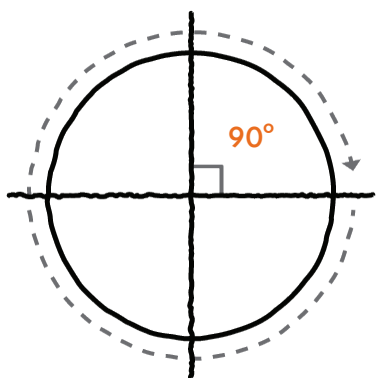
Repeat the code between the 2 brackets 4 times

To run the program, push the ‘run’ button in the toolbar. The program should draw the rectangle all at once.

Drawing a Circle

Drawing a circle on a computer is different from drawing a circle by hand. A computer can only draw straight lines, but we can create the *appearance* of a curved line by drawing a bunch of tiny lines that, when connected together, look like a circle.

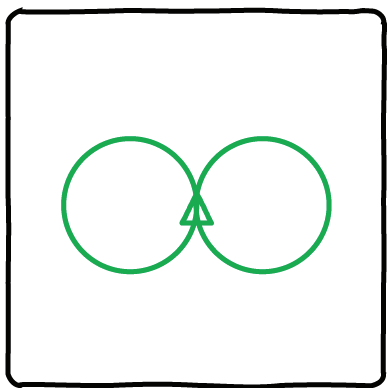
Since a circle has 360 degrees, we can create a circle by making 360 tiny lines, each one pixel long. After each line is drawn, we turn the pen 1 degree to the right.



Type the following and then press the “run” button:

```
clrturtle  
home  
repeat 360 [  
    forward 1  
    right 1  
]
```

What if you wanted to draw a circle to the left? You could repeat the same commands except change the command “right” to the command “left.”



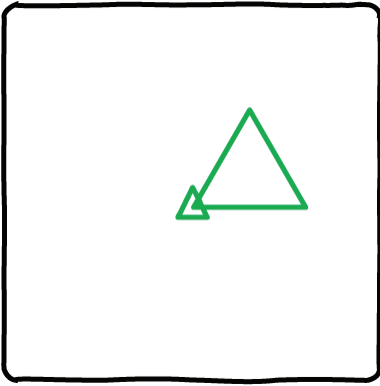
```
clrturtle
home
repeat 360 [
    forward 1
    right 1
]
repeat 360 [
    forward 1
    left 1
]
```

Hint: To make the circle bigger we can change the length of line (For example: forward 3).

Drawing a Triangle

We can also use the repeat command to easily make an equilateral triangle (a triangle in which the length of all three sides are equal). Since each of the internal angles

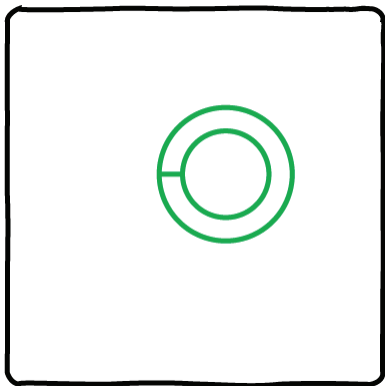
must also be equal we can create a triangle by rotating the pen 120 degrees after each line is drawn.



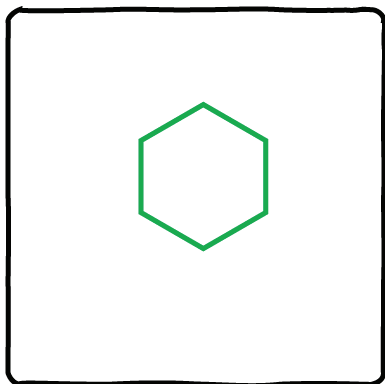
```
clrturtle  
home  
  
repeat 3 [  
    forward 80  
    right 120  
]
```

Ch 5: Practice Problems

1) Can you draw a small circle nested inside a big circle?



2) Try drawing a hexagon (a shape with 6 sides). To figure out the angles between the lines, divide 360 degrees by the number of sides.

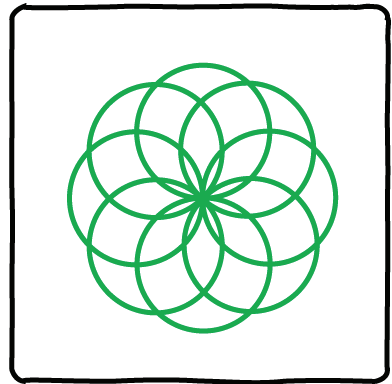
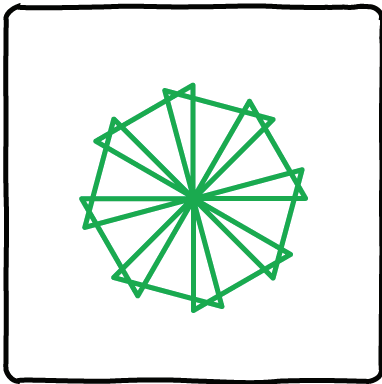


Nested Repeats

06

The really cool thing about repeat is that it allows us to draw the same shape as many times as we want with only a few lines of code.

In the last chapter we learned about using the repeat command. We can also **nest** a repeat command inside another repeat command. This allows us to build more complex shapes.



Nested loops can look a little complicated the first time you see them, so let's go through the code line by line for the drawing on the left.

```

clrturtle
home
repeat 8 [
  repeat 3 [
    forward 100
    right 120
  ]
  right 45
]

```

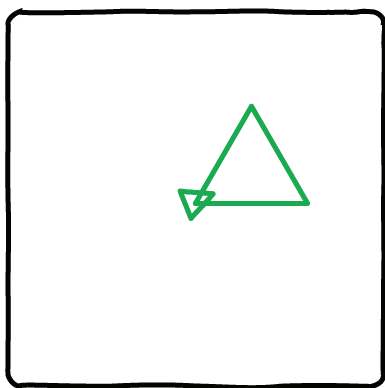
Repeat the code between these 2 brackets 8 times

Repeat the code between these 2 brackets 3 times

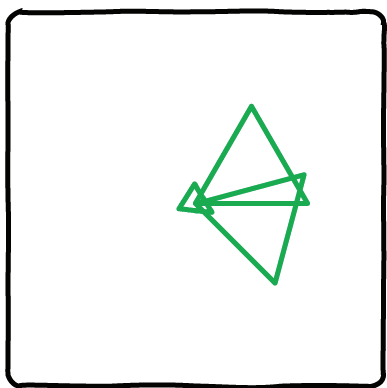
Step 1) First, MLogo does the inner repeat command `repeat 3[forward 100 right 120]`. This creates a triangle.

Step 2) MLogo then does the `right 45`. This turns the pen right by 45 degrees.

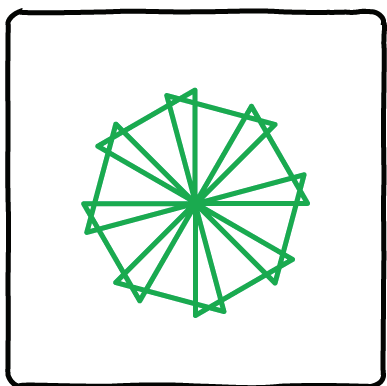
Step 3) After these two steps, MLogo has drawn a triangle. MLogo is now also ready to draw the next triangle. The screen looks like this:



Step 4) MLogo then does steps 1 and 2 again. The screen now looks like this:

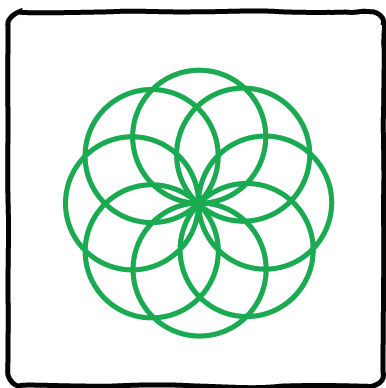


Step 5) MLogo continues to do steps 1 and 2 until it has done them 8 times. After the 8th time, MLogo stops. The final screen now looks like this:



Nested Repeats with Different Shapes

We can make lots of interesting shapes by repeating a simple shape over and over again. Other shapes that we could use as a base are circles, pentagons (shapes with 5 sides), octagons (shapes with 8 sides).



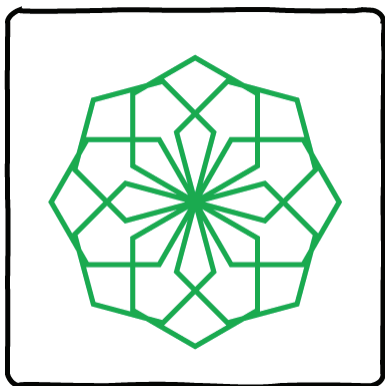
To make the shape above, a circle was drawn 8 different times. After each circle was drawn, the pen was rotated 45 degrees to the right. The code for the shape is as follows:

```
clrturtle  
home  
repeat 8 [  
    right 45  
    repeat 36 [  
        forward 10  
        right 10  
    ]  
]
```

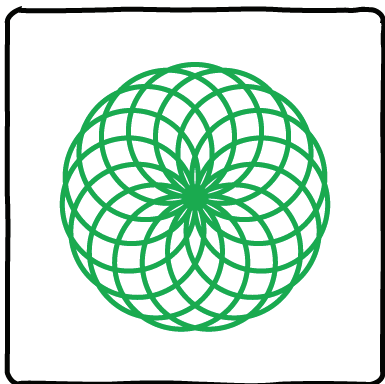
What would happen if you repeated the outer loop 36 times? Or if you changed the angle the pen rotates from 45 degrees to 10 degrees?

Ch 6: Practice Problems

1) Try to draw the shape below, which is just a repeat of a hexagon shape (a shape with 6 sides).



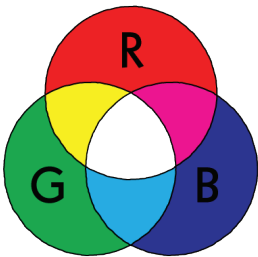
2) Edit the nested code for the circle shape in this chapter so there are more circles in the shape.



Making Colors

07

So far we have only been able to draw with the color green, but we can change the color of the pen to any color that we want.














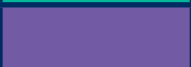

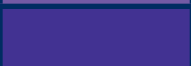
Remember from art class, if you wanted to create orange paint you would mix yellow and red paints together. Creating colors on the computer is a little bit different, because when computers create colors they mix light, not pigments.

RGB Three colors of light: red, blue and green, can be mixed to create pretty much every color imaginable. The red, blue, and green recipe is known as a **RGB value**. The RGB color value is broken into three numbers, each of which is between the values 0 and 255.

The Code To change the color of the pen, put the following code before the line that you want to change:

```
setcolor(red blue green)
```

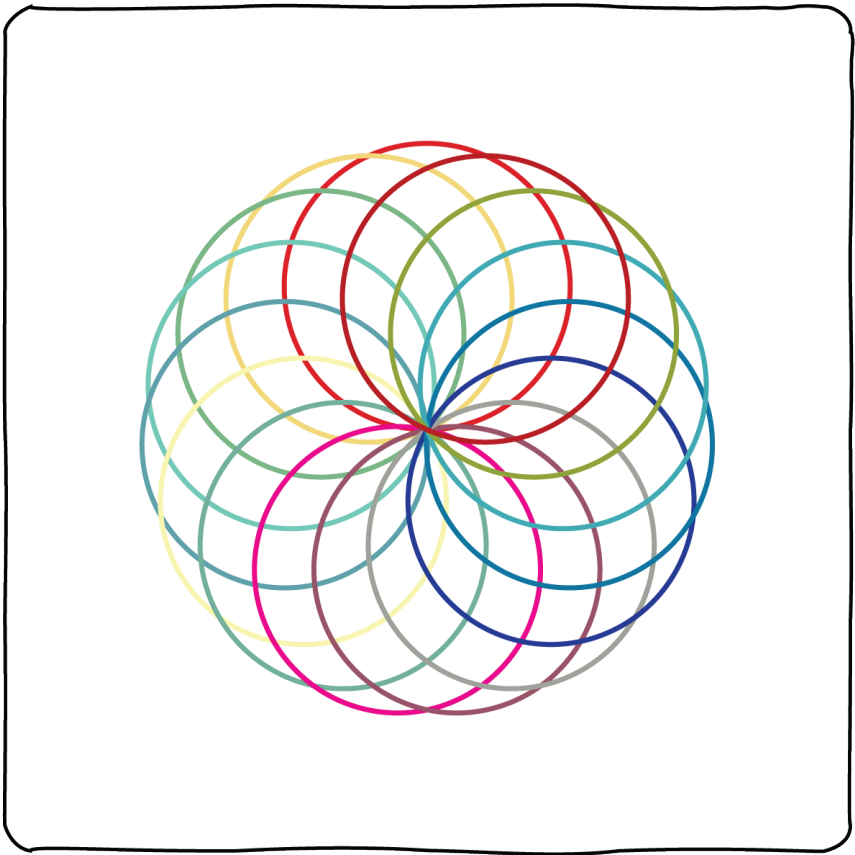

Finding RGB Values The easiest way to create a color is to find a color you like in a color palette and get the RGB value. I've provided a chart of some common RGB color values, but you can also find RGB color charts with hundreds of colors on the internet.

Common Colors			
Color	RGB Value	Color	RGB Value
	(0, 0, 0)		(255, 233, 56)
	(255, 255, 255)		(232, 109, 31)
	(174, 224, 232)		(141, 200, 86)
	(0, 161, 255)		(0, 169, 79)
	(0, 0, 223)		(14, 183, 154)
	(229, 14, 99)		(115, 89, 166)
	(210, 23, 42)		(67, 48, 146)

How To Use the Code In the code example below, the color is set to red before the rectangle is drawn. Notice that the values in setcolor are the same as the RGB values for red in the table above.

```
setcolor(210 23 42) ← Set the color
repeat 4 [
  right 90
  forward 100
]
```

Random Colors To create random colors, we need to generate random values for each of the red, blue, and green values in setcolor.



Random The code `random(255)` generates a random number between 0 and 255. We chose 255 as the upper bound because an RGB value can only be between 0 and 255.

The code below generates a new color every time one circle is drawn.

```
clrturtle  
home
```

```
repeat 15 [  
    setcolor(random(255) random(255) random(255))  
    right 24  
    repeat 36 [  
        forward 10  
        right 10  
    ]  
]
```

All About Variables

08

What is a Variable? Sometimes we want to be able to keep track of a value that might change as the code executes. In order to keep track of that value, we need to store it memory so we can access it again. **Variables** are a way for programmers to store values. It is helpful to think of a variable as a box that can only store one value. When a new value is put in the box, the old value is destroyed forever.

How to Create a Variable When we create a variable, we are creating a storage space in memory to store a single value. To create a variable, we must first use the word “**make**” to tell MLogo that we are declaring a variable. We then give the variable a **name**, which must start with an apostrophe, and finally, we assign a value to the variable. Storing a value in a variable for the first time is called **initializing the variable**.

The diagram shows the MLogo command `make 'newVar 5` inside a light blue rounded rectangle. Three orange arrows point from text labels to parts of the command: one from “make” to the word `make`, one from “the variable name” to the variable name `'newVar`, and one from “the variable is initialized to 5” to the value `5`.

“make” tells MLogo that we are declaring a variable

`make 'newVar 5`

the variable name

the variable is initialized to 5

Variable Name Programs can have lots of variables, so we need a way to tell the variables apart. The way we do this is by assigning the variable a name. There are a few rules you need to observe when making a variable name.

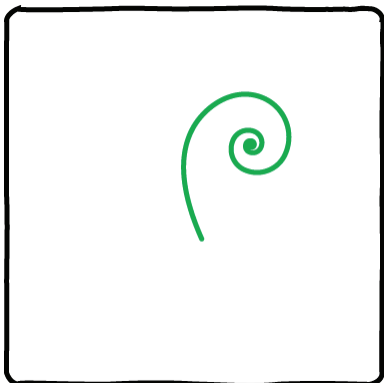
- 1) Always begin your variable name with a apostrophe.
- 2) There should be no spaces in the name. Programmers use **camel case** to make variable names easier to read. The first word is lowercase and each successive word has its first letter capitalized (for example: myFirstVariable, aVeryLongVariableName, redColor).
- 3) Variable names are case sensitive, so the variable name finalScore is not the same as finalscore.

Changing the Value of a Variable At some point we will want to change the value stored in a variable. We can set the value to a specific number or we can perform some math on the variable already being stored.

Doing math is a little bit different in MLogo than what you remember from math class. Instead of using a symbol like + for addition, we use the word “sum.”

Math Commands		
Command	Math	Command In MLogo
add	$2 + 4$	sum:2 4
subtract	$5 + 9$	dif:5 9
multiply	$2 * 2$	product:2 2
divide	$8 / 4$	quotient:2 4

Project: Make a Spiral



When we draw a spiral, we need to adjust the angle of pen after each line segment is drawn. Now that we have learned about variables, it will be easy for us to draw a spiral because we can store the angle value in a variable.

```
home
clrturtle
make 'angle 0
repeat 36 [
  forward 10
  right :angle
  make 'angle sum:angle 2
]
```

← the variable 'angle is initialized to 0

← get the value in 'angle

← Every time we get to the end of the repeat loop, 2 will be added to the current value of 'angle

How variables work The first time we see the word “make,” we are declaring and initializing the ‘angle variable. The second time we see make, we are updating the value held in the ‘angle variable. The computation takes the current value held by the variable “angle,” adds 2 to it, and then puts the new value back into “angle.”

When we first start the program, the value stored in ‘angle is 0. As the program progresses, the variable ‘angle will store the values 0, 2, 4, 6, 8, 10, and so forth.

Getting the value of a variable To get the value in the variable we have to use a semicolon in front of the variable name like so `:angle`.

Ch 8: Practice Problems

- 1)** True or False. Are these two variable names the same: 'myFirstVariable' and 'MYFIRSTVARIABLE'?
- 2)** What is the final value stored in x after the program executes?

```
make 'x 7  
make 'x sum:x 5
```

The final value of x is: _____

- 3)** What is the final value stored in y after the program executes?

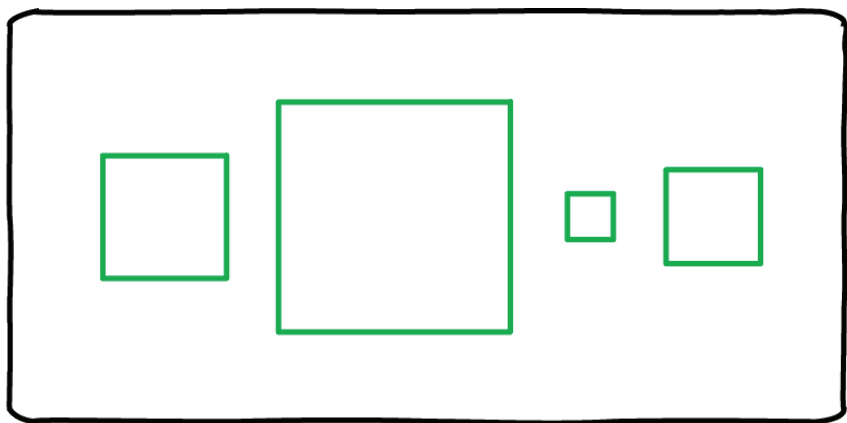
```
make 'x 2  
make 'x product:x 5
```

The final value of y is: _____

Awesome Functions

09

Why Use Functions? When we used the repeat command, it made drawing complex shapes really easy. Now we will learn about functions, which allow us to repeat the same code over and over again without having to rewrite the code. A function may sound an awful lot like the repeat command, but functions are one of the most useful tools a programmer can have because it allows us to adjust certain parts of the code based on our needs.



The limitation of the repeat command is that if we want to draw a bunch of squares, each with a different side length, each square will be coded with virtually identical code

except for the length of the sides. With a function, we can use one block of code to draw different size squares!

Formatting Functions Functions have to be written in a particular way. The format of a function is as follows:

```
to square :size
  repeat 4 [
    forward :size
    right 90
  ]
end
```

to means that the code between “to” and “end” is a function

square is the name of the function

:size says the command will have an input and in the function, its name will be “:size.” All inputs must have a colon(:) in front of the name. A function can have multiple inputs.

end marks the end of the function code

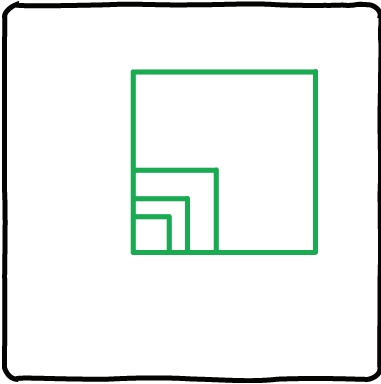
Calling A Function If you type the code above into the text editor of MLogo and click “run” nothing will happen. That is because we have to tell the computer to call the function. The way we call the function above is by adding the following code after the function

square 10 ← the input value

↑
the function name

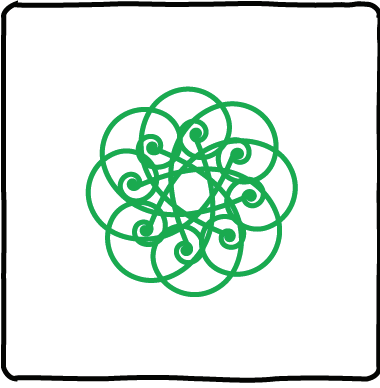
Project: Drawing Squares

Let's use a function to draw a bunch of different size squares. Can you draw some other size squares?



```
to square :size
  repeat 4 [
    forward :size
    right 90
  ]
end
square 10
square 100
square 20
square 30
```

Project: Spiral Rose

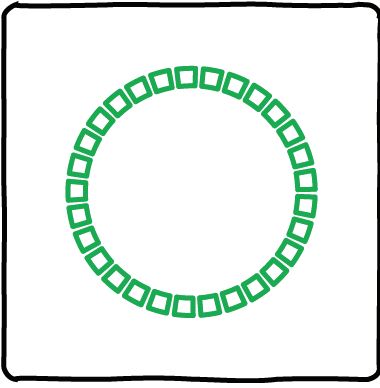


We can combine repeats and functions to make the spiral rose above.

```
to spiral :length
  make 'angle 0
  repeat 36 [
    forward :length
    right :angle
    make 'angle sum:angle 2
  ]
end

home
clrturtle
repeat 8 [
  right 45
  spiral 10
  forward 20
]
```

Project: Circle of Squares



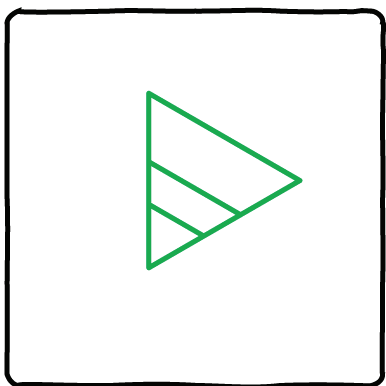
Lots of fun things can be created with functions like a circle made out of squares. The command `penup` is used to stop the pen from making marks while it is moving.

```
to circleOfSquares :size
  pendown
  repeat 4 [
    forward :size
    right 90
  ]
  penup
end

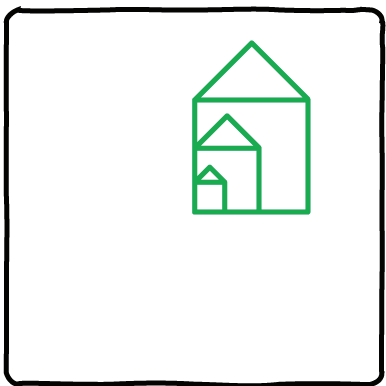
home
clrturtle
penup
repeat 38 [
  forward 15
  circleOfSquares 10
  right 10
]
```

Ch 9: Practice Problems

1) Can you make a function that will help you draw different size triangles?



2) Make a function that will let you draw different size houses.

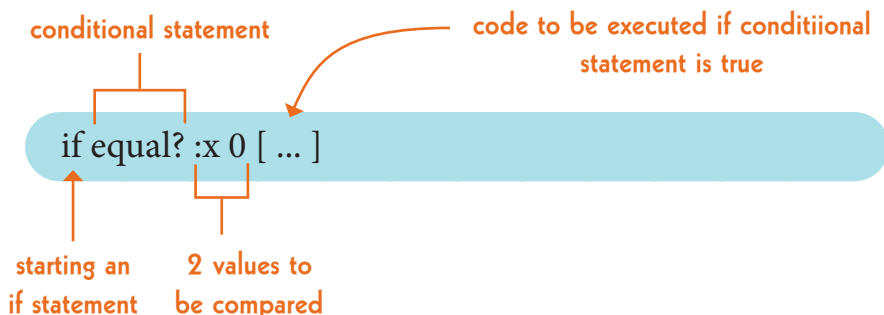


If Statements

10

What if you wanted your pen to do different things based on the current value of a variable? If the variable's value is greater than zero, the pen should rotate 45 degrees. If the variable's value is equal to zero, the pen should rotate 90 degrees. An easy way to program this is to use an if statement.

If Statement The if statement tells your program to execute a certain section of code only if a particular condition is true. If it is true, the code between the brackets will be executed. If it is false, MLogo will simply ignore the code between the brackets.

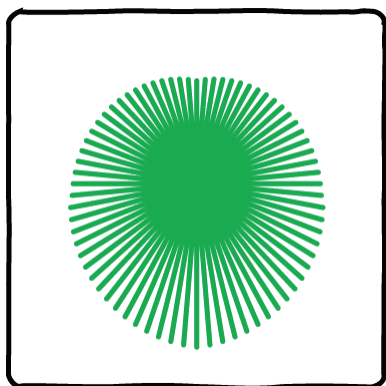


Conditional Statements MLogo lets you compare values to see if one is greater than the other, less than the other or equal to the other value. Instead of using symbols like $<$, $>$, and $=$, we use `less?`, `greater?`, and `equal?` to compare things in MLogo.

Conditional Statements		
Command	Example	Command In MLogo
equal	$5 = 6$	<code>equal? 5 6</code>
less	$10 < 3$	<code>less? 10 3</code>
greater	$8 > 5$	<code>greater? 8 5</code>

Hint: There should be no space between the word and the question mark in `less?`, `greater?`, and `equal?`. MLogo will refuse to run if the syntax is not correct.

Project: Radiating Lines



To make the image to the left, we used two if statements to adjust the length of the lines.

The repeat command is going to run 73 times. We are going to use a variable called 'count' to keep track of what number repeat we are on.


```
to line :length :angle  
  right :angle  
  forward :length  
end
```

line function

```
clrturtle  
home  
penup  
forward 100  
pendown
```

move the pen so
the whole drawing
will appear on the page

```
make 'count 0  
make 'lineLength 150  
make 'angle 180
```

initialize and declare 3
variables

```
repeat 73 [  
  home  
  line :lineLength :angle  
  if less? :count 36 [  
    make 'lineLength dif:lineLength 2  
  ]  
  if greater? :count 36 [  
    make 'lineLength sum:lineLength 2  
  ]  
  make 'angle dif:angle 5  
  make 'count sum:count 1  
]
```

return the pen to the middle of the screen

call the line function

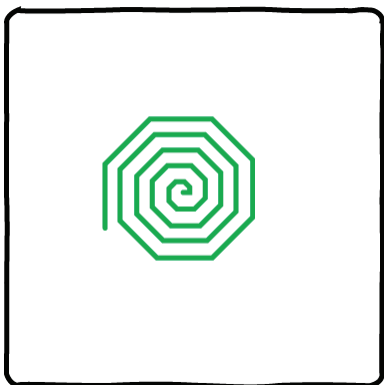
angle increases by 5

count increases by 1

if the count is less than 36, increase the line length

if the count is greater than 36, decrease the line length

Project: Recursive Spiral



An interesting thing about functions is that they can call themselves. Whenever a function calls itself, it is called **recursion**. The only problem with this technique is that we need a way to stop the calling process or else the program will run forever!

If loops are helpful for stopping the program because we can tell the program to stop once a certain condition has been met. For example, we could initially call the loop with the variable 'x' that has been initialized to 100. Each time the loop calls itself, the variable decreases in size by 1. When the value of 'x' reaches 0, the program will stop.

```
to spiral :x
  if equal? :x 0 [
    stop
  ]
  forward :x
  right 45
  make 'x dif :x 1
  spiral :x
end

home
clrturtle
spiral 50
```

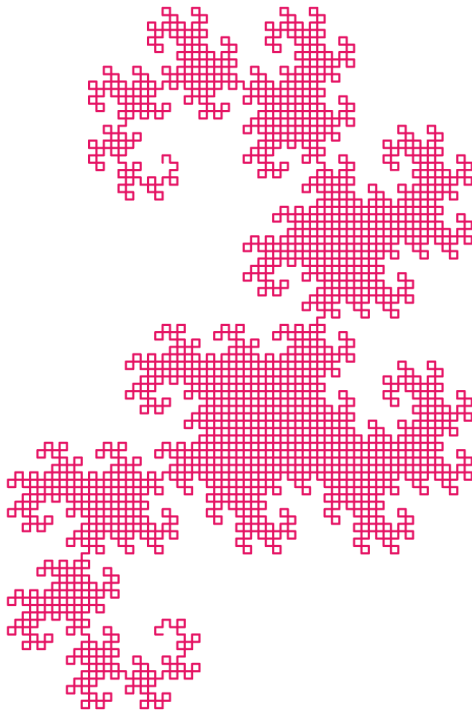
if the value held in 'x equals 0, stop the program

decrease the value in x by 1

the spiral function calls itself with x

initial call to the spiral function

Project: Dragon Curve



Programmers like recursion a lot because it lets them make really complicated drawings using relatively few lines of code. If we tried to draw the dragon curve above only by using functions and repeat loops, it would take us hours, and hundreds of lines of code, to complete.

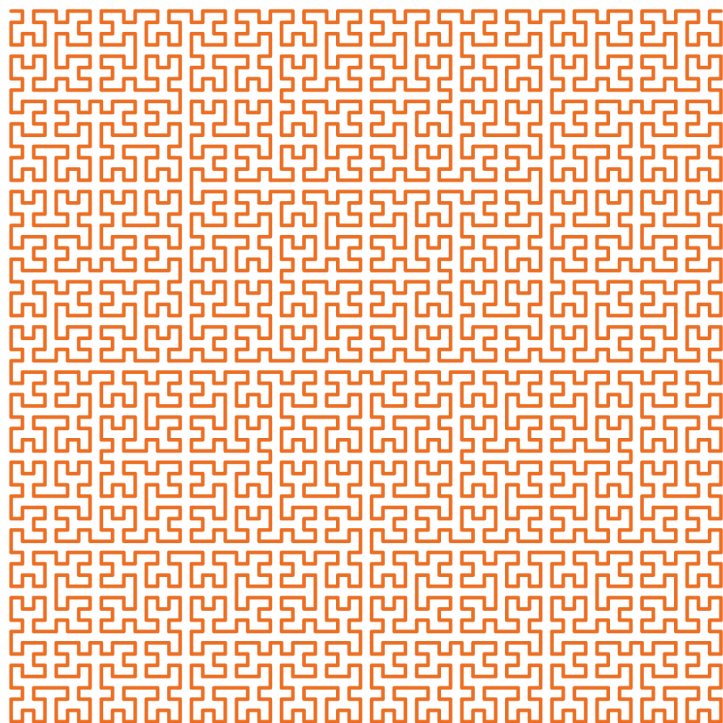
Try experimenting with the code below by calling the function with different inputs (for example call the function `x` on the last line of code with `x 6` or `x 13`).

```
to x :c
  if equal? :c 0 [stop]
  x dif :c 1
  right 90
  y dif :c 1
  forward 4
end
```

```
to y :c
  if equal? :c 0 [stop]
  forward 4
  x dif :c 1
  left 90
  y dif :c 1
end
```

```
home
clrturtle
setcolor(229 14 99)
penup
forward 100
pendown
x 10
```

Project: Hilbert Curve



The labyrinth above is called the Hilbert curve. Like the dragon curve, it is really easy to draw with a recursive function.

After you copy the code into MLogo and run the program, try experimenting with altering the code. For instance, try changing the inputs for the LSec function call. For example change the inputs in the last line of code to `LSec 5 5` or `LSec 3 3`).

```
to LSec :h :w
  if equal? :h 0 [stop]
  right 90
  RSec dif :h 1 :w
  forward :w
  left 90
  LSec dif :h 1 :w
  forward :w
  LSec dif :h 1 :w
  left 90
  forward :w
  RSec dif :h 1 :w
  right 90
end
```

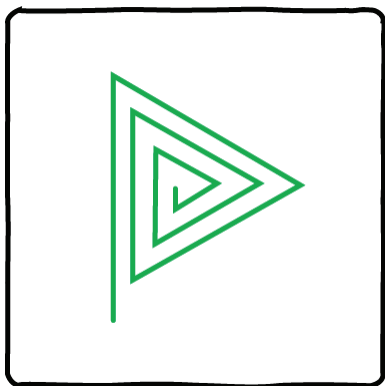
```
to RSec :h :w
  if equal? :h 0 [stop]
  left 90
  LSec dif :h 1 :w
  forward :w
  right 90
  RSec dif :h 1 :w
  forward :w
  RSec dif :h 1 :w
  right 90
  forward :w
  LSec dif :h 1 :w
  left 90
end
```

```
clrturtle
home
penup
left 90
forward 150
left 90
forward 150
right 180
```

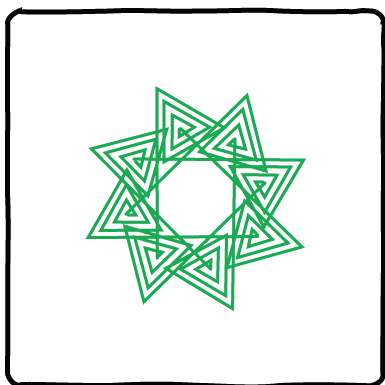
```
pendown  
fast  
setcolor(232 109 31)  
LSec 6 6
```

Ch 10: Practice Problems

1) Can you make a recursive spiral triangle?



2) Use the spiral triangle code from above to make a star. Experiment with different angles and lengths to create different types of stars.



Words & Printing

11

So far we have only used MLogo to draw lines and shapes, but we can also manipulate words with MLogo.

Printing Words We can use the print function to print a word to the History section of the MLogo IDE (page 6 has a diagram of the MLogo IDE).

```
to write
  print 'Hello
  print 'World
end

write
```

← we need to use an apostrophe in front of the word to let MLogo know that it is a word

When we run the program, you will see the following output in the History Section of the IDE:

```
>'Hello
>'World
```

Printing on One Line You may have noticed that the two words we printed out are on separate lines. To print them on the same line we have to use the following format:

```
to write
  print
  fput 'Hello
  fput 'World
  []
end

write
```

You will see the following printed out when you run the program:

```
>['Hello 'World]
```

Arrays To learn how fput works, first we need to learn about arrays. Remember when we learned about variables? Variables are a way to store one thing in memory. Sometimes we need to store multiple things in memory but we don't want to do a lot of tedious typing. Arrays let us store many variables in a list-like structure.

fput We know we are dealing with an array when we see the an opening and closing bracket together ([]). What fput is doing is adding each word to the end of the empty array. The print function then prints out the array.

Project: Silly Poem

We are going to create a randomly generated poem that will output something different everytime we click the run button.

```
to any :list
  output item random size :list :list
end
```

```
to noun
  output any [
    [an enraged camel]
    [an ancient philosopher]
    [a cocker spaniel]
    [the Eiffel Tower]
    [a cowardly moose]
    [the silent majority]
  ]
end
```

```
to verb
  output any [
    [get inspiration from]
    [redecorate]
    [become an obsession of]
    [make a salt lick out of]
    [buy an interest in]
  ]
end
```

```
to object
  output any [
    [mother in law]
    [psychoanalyst]
    [rumpus room]
  ]
end
```

```
        [fern]
        [garage]
        [love letters]
    ]
end
```

```
to curse
    print fput 'May
    fput noun
    fput verb
    fput 'your
    fput object
    fput '!'
    []
end
```

```
curse
```

Ch 11: Practice Problems

1) Try editing the poem to make different word combinations.

Problem Solutions

01

2: First Program

1) There are many ways to draw a house. The code below shows one way to draw a house by first drawing the rectangle and then adding a triangle to the top of the image.

```
right 90  
forward 100  
right 90  
forward 100  
right 90  
forward 100  
right 90  
forward 100  
right 45  
forward 70  
right 90  
forward 70
```

2) A diamond can be drawn in many ways. The code below shows a simple way to draw a diamond.

```
right 45
forward 100
right 90
forward 100
right 90
forward 100
right 90
forward 100
```

5: Repeat Command

1) There are many ways to draw different size circles. The code below shows one way to draw two different size circles. The first repeat code draws the smaller inner circle and the second repeat code draws the bigger outer circle.

```
clrturtle
home
repeat 120 [
    forward 2
    right 3
]

right 90
back 18
left 90

repeat 360 [
    forward 1
    right 1
]
```

2) The easiest way to draw a hexagon is to draw 6 lines with an angle of 60 degrees between each line.

```
clrturtle  
home  
  
repeat 6 [  
    right 60  
    forward 60  
]
```

6: Nested Repeats

1) There are many ways to draw a shape consisting of hexagons. The code below describes one way to draw the shape.

```
clrturtle  
home  
  
repeat 8 [  
    repeat 6 [  
        right 60  
        forward 60  
    ]  
    right 45  
]
```

2) The code below describes one way to draw a shape made out of many circles

```
clrturtle  
home  
  
repeat 15 [  
    right 60  
    forward 60  
]
```



```
right 24
repeat 36 [
  forward 10
  right 10
]
```

8: All About Variables

1) False. since the computer thinks lower and upper case letters are different, 'myFirstVariable' and 'MYFIRST-VARIABLE' do not mean the same thing to the computer.

2)

The final value of x is: 12

3)

The final value of y is: 10

9: Awesome Functions

1) There are many ways create a function that draws triangles. Below is one way to do it.

```
clrturtle
home
to triangle :length
  repeat 3 [
    forward :length
    right 120
  ]
end
```

```
triangle 50  
triangle 100  
triangle 70
```

2) The code below shows you one way to draw a resizable house.

```
to house :length  
  repeat 4 [  
    forward :length  
    right 90  
  ]  
  
  forward :length  
  right 30  
  
  repeat 3 [  
    forward :length  
    right 120  
  ]  
end  
  
clrturtle  
home  
house 10  
home  
house 30  
home  
house 60
```

10: If Statements

1) There are many ways code a recursive triangle. Below is one way to do it.

```
to triangleSpiral :x
  setcolor(255 0 0)
  if equal? :x 0 [
    stop
  ]
  forward :x
  right 120
  make 'x dif :x 10
  triangleSpiral :x
end

home
clrturtle
triangleSpiral 100
```

2) To create the star in the picture provided, copy this code:

```
to triangleSpiral :x
  setcolor(255 0 0)
  if equal? :x 0 [
    stop
  ]
  forward :x
  right 120
  make 'x dif :x 10
  triangleSpiral :x
end
```

```
home
clrturtle
fast
repeat 8 [
    triangleSpiral 100
    right 15
    forward 100
]
```


Name:

Email:

Text and Illustrations By: Nicole Yarroch

© 2014 Nicole Yarroch