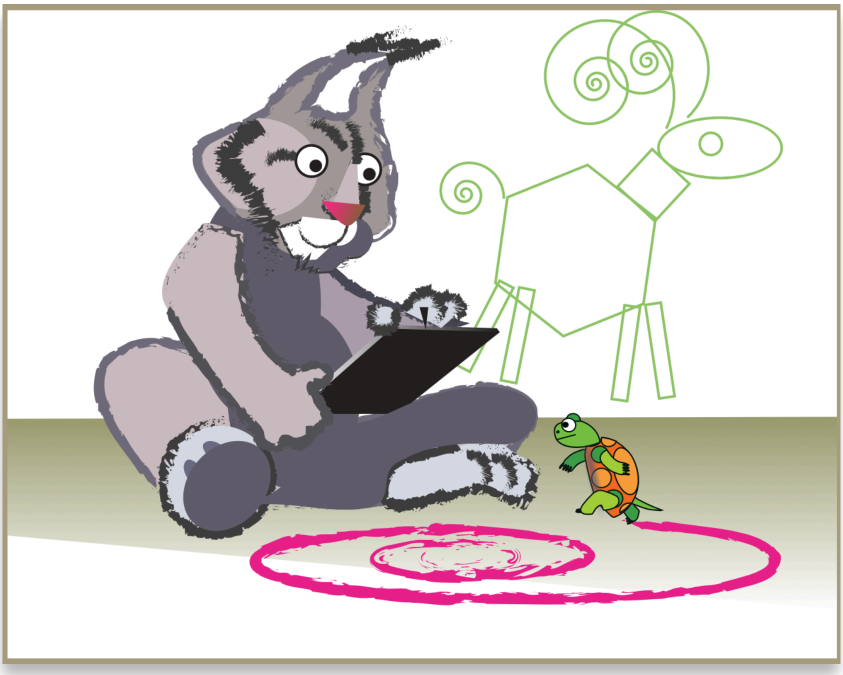
Project 2 - The art of the lynx

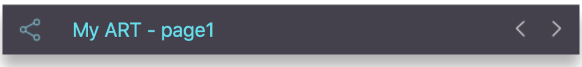


Project 2 starts where Project 1 ended. You are going to make more geometric figures, this time much more efficiently, and use this extra power to create geometrical and random art. We will introduce **procedures with inputs** (**variables**), **super** and **sub-procedures**, `random`, `forever`, **conditional statements**, `stop`, and **buttons**.

Start a new project from the Lynx home page, or from your **My Projects** page. If you are already inside the Lynx editor, you can choose **New project** from the **Down from the cloud** button:

If there isn't a turtle on the page, choose **Turtle** in the + menu.

As a good habit, start by **naming your project**. Use a name that makes sense for your project. You don't want 10 projects called **My Projects.**


My ART - page1

And again, **save your project often**. Simply click on this icon:

Available in many sizes

In Project 1, if you wanted small, medium and large squares, you had to create *three* procedures, with different names, one for each size. There is a much better way to do this.

Open the **Procedures Pane** and create this procedure:

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

Now try to run `square` from the Command Centre:

```
square
square needs more inputs in square
```

This is an **error message**. You see, `square` now works like `forward`. It also **NEEDS** a number to run. That number is called an **input**, or a **variable** (like in math). Try this in the Command Centre:

```
pendown
square 10
square 50
square 100
```

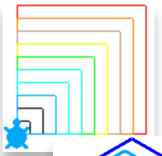
IMPORTANT INFO!

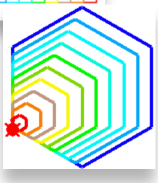
To create a procedure with an input, add the input on the title line, preceded by a colon (`:size`). It can be any name but must be a single word. No space between the colon (:) and the name (size)!

Next, you must use that same word (`:size`), again with the colon, inside your procedure. If you run `square 100`, the variable `:size` will work using the value 100.

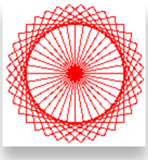
Can you make this pattern using your new `square` procedure?

Can you create another polygon procedure with an input, to create polygons of any size?





Now that's a SUPER procedure!



Remember the "squares" code from Project 1? This challenge can be easier and more fun with the following technique:

You already have a `square :size` procedure (see previous page). Now create this one:

```
to manysquares :size
repeat 36 [square :size right 10]
end
```

What is going on here… The procedure `manysquares` **USES** the procedure `square`. In this case, you can say that `manysquares` is a **SUPER procedure**, and `square`, because it is used inside `manysquares`, is called a **SUB-procedure**.

Clear the graphics and type this in the Command Centre:

```
setcolour 15
manysquares 80
```

Did you expect this?

Again… can you figure out why `right 10` is included? Remember the Total Turtle Trip?

Introducing… random

`Random` does only one thing. It returns a **random number**. It is like choosing a number by chance.Try this in the Command Centre:

```
cc
repeat 100 [show random 80]
44
6
62
37...
```

THIS MEANS CLEAR THE COMMAND CENTRE

YOU WILL GET A HUNDRED NUMBERS LIKE THESE. YOUR NUMBERS WILL CERTAINLY BE DIFFERENT. YOU MAY GET THE SAME NUMBER A FEW TIMES.

In this case, `random 80` returns numbers between **0** and **79**. You can use that feature to make random art.

`Random` ***`number`*** returns a random number between 0 and that ***`number`*** minus one. `Random 100` returns a number between 0 and 99.

Try this **a few times** in the Command Centre:

```
square random 100
```

Note: If the random number is very small, you may get a square that is too small for you to see because the turtle is covering it. Run `square random 100` again.

Now try this. You know what `manysquares 80` does, but what about this instruction?

```
manysquares random 80
```

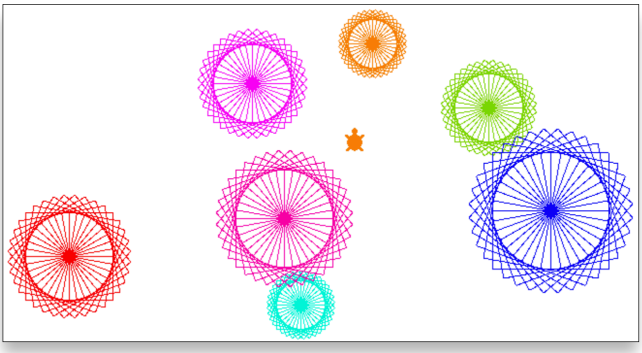
Yes, Lynx chooses a **random size** to be given to `forward`. Let's add **random colours** to the recipe.

So far, you have been using colour names with the command `setcolour` (like `setcolour 'red'`). You can also use numbers! Take a look at the colour chart in *Appendix C*. There are 140 colours and each has a specific number.

Drag the turtle some other place on the page, and run these two lines*:*

```
setcolour random 140
manysquares random 80
```

Keep moving the turtle and run the same 2 lines a few times. Do you get something like this?



If you drag the turtle around, remember that you can use `penup home pendown` to bring it back to the centre of the page.

Project 2 - The art of the lynx

All together now!

Using all the features above (variables, random, super procedure), you can edit the `manysquares` procedure so it calls not *one*, but *two* sub-procedures:

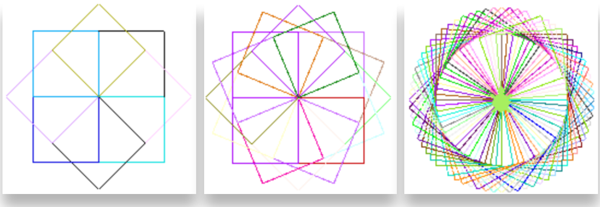
```
to manysquares :size
repeat 36 [changecolour square :size right 10]
end
```

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

```
to changecolour
setcolour random 140
end
```

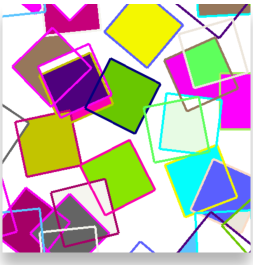
In the Command Centre type `manysquares random 100`. This instruction uses the sub-procedure `square` and a random number for the size, and it uses the sub-procedure `changecolour` to choose a random colour.

Consider these other styles of "`manysquares`". Can you figure out what to change in the procedures? Make your own!



I can go on forever like that

Look at the pattern on the right. Try to understand or "deconstruct" what you see, try to describe, in your own words, what you see and how this was done.



Use the words `random`, `square`, `forward`, `right`, `setcolour` and `fill` in your description.

SPOILER ALERT: Work on this challenge in your head. Don't look at the next page yet.

Your description could be something like: turn right a random amount, move forward a random amount, draw a square, pick a random colour, go inside the square and fill it… Do that forever.

You already have a **square** procedure. Create a procedure to go *inside* the square and paint it with a random colour. Also create a procedure to move to a random place on the page. Notice the spelling of **paint.inside**. It is still just *one word, with a dot, but without a space*, because a procedure name **has** to be **one word**. Instead of a dot, you could use an underscore (_) to link the two words.

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

to paint.inside
; pu before going inside and pd before filling
pu rt 20 fd 20 pd
setcolour random 140
setpensize 2     ; so fill doesn't leak out
fill
end

to move
; must use penup before moving away
; then pendown when turtle arrives at new location
; random 360 goes in ANY direction (0 to 359)
pu rt random 360
fd random 300 pd
end
```

Try all these procedures one by one in the Command Centre: `square 100`   `paint.inside`   `move`. Trying them individually will reveal bugs, if any.

Then, create a super procedure that moves, makes a square, and paints it FOREVER!

```
to pattern
setpensize 4
forever [move square 80 paint.inside]
end
```

Did you notice the plain English comments in the procedures above? Any line that starts with a semi-colon (;), inside or outside of a procedure, is a COMMENT.

Comments are a sign of good coders as they help you, and others, understand what you are trying to do!

Project 2 - The art of the lynx

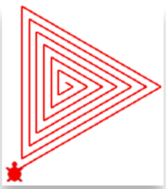
Launch the procedure `pattern` from the Command Centre.

```
pattern
```

The `forever` primitive does exactly what you expect. You will need this tool (the **stopall** button, to the left of the Command Centre) to stop the action.

Spirals - I have my conditions!

Talking about going "forever"… Here's a solution to the problem "forever may be too much".



Describe this in your own words, but DON'T use the word "spiral":

Your words might be something like: draw a line, turn, draw a longer line, turn, draw a longer line… and so on.

This procedure starts doing the job: it draws a line once and turns at an angle once. The procedure has two inputs (variables), one for the `:size`, one for the `:angle`

```
to spiral :size :angle
forward :size
right :angle
end
```

Try this `spiral` procedure in the Command Centre. For the moment, it does only one line at a time, so you will have to run it many times. Use `cg` between each try.

```
cg setpensize 1 pd
spiral 20 120
spiral 30 120
spiral 40 120
spiral 50 120
```

You see where this is going? There must be a better way!

Let's add some magic. One line is added to the procedure: the procedure **CALLS ITSELF**, this time with a longer line (`:size + 10`), and the same `:angle`.

```
to spiral :size :angle
forward :size
right :angle
spiral :size + 10 :angle
end
```

A procedure that calls itself is called a RECURSIVE procedure. It is extremely powerful.

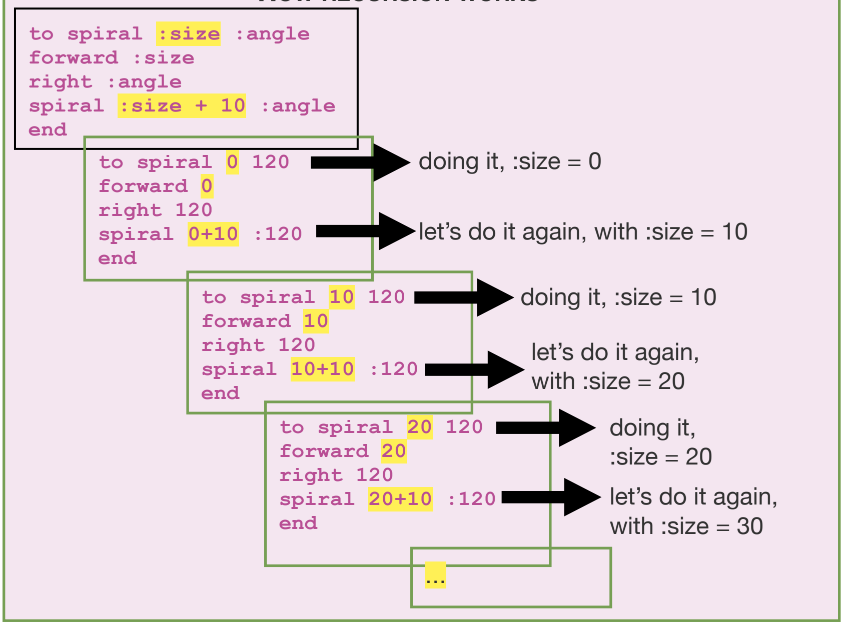
Type this in the Command Centre.

```
spiral 0 120
```

Nice for a short while, but it goes overboard pretty fast, and you get an error message. Use this to stop the procedure:

Why, or how, is it growing? Examine this special line (the recursive call):

HOW RECURSION WORKS



```
to spiral :size :angle
forward :size
right :angle
spiral :size + 10 :angle
end
```

```
to spiral 0 120
forward 0
right 120
spiral 0+10 :120
end
```

```
to spiral 10 120
forward 10
right 120
spiral 10+10 :120
end
```

```
to spiral 20 120
forward 20
right 120
spiral 20+10 :120
end
```

```
...
```

doing it, :size = 0
let's do it again, with :size = 10
doing it, :size = 10
let's do it again, with :size = 20
doing it, :size = 20
let's do it again, with :size = 30

Let's make this procedure **stop by itself** before the spiral grows too large. Add this new line to the `spiral` procedure:

```
to spiral :size :angle
if :size > 200 [stop]
forward :size
right :angle
spiral :size + 10 :angle
end
```

The "`if`" instruction is a STOP RULE, or a CONDITIONAL STATEMENT. You need this to stop recursive procedures.

How IF works

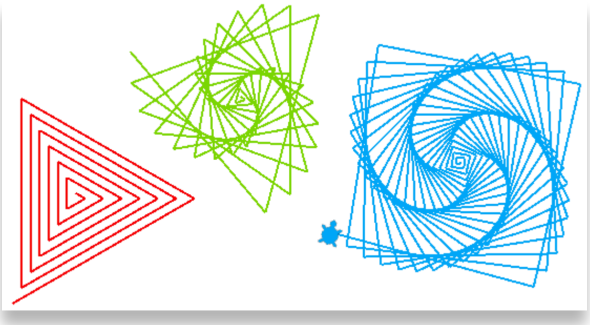
```
IF :SIZE > 200 [STOP]
```

```
IF THIS-IS-TRUE THEN-DO-THIS
```

Again, try these, and some of your own, from the Command Centre, use cg between each try.

```
spiral 0 120
spiral 0 125
spiral 0 115
spiral 0 90
spiral 0 95
```

What is happening here?
And here - can you explain this?



Can you modify the procedure to change the **maximum size** of the spiral (smaller, larger)?

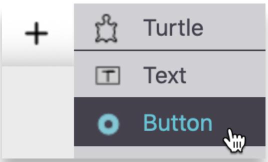
Can you modify the procedure so the spiral **grows faster or slower** (the amount that is added at each arm)?

Click me!

So far, you have been using the Command Centre to “try” things, and that’s exactly what it is for. Soon, you will **share** your projects or use them **without** a Command Centre. You need a way, **INSIDE THE PAGE**, to trigger the action.

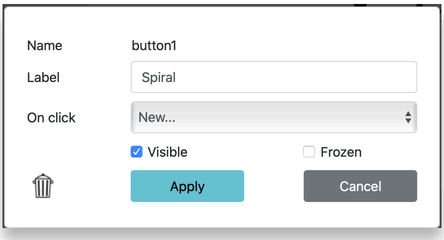
Introducing **buttons**. Imagine just dragging a turtle to a new location, and just clicking on a button to get a spiral, right on that spot.

Click on the + menu and choose **Button**.



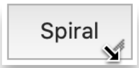
A button, named **nothing** by default, appears in the centre of the page. A button is just a visible object that runs code when clicked. Right-click on the button to open its dialog box.

In the dialog box that appears, type **Spiral** as the label (it can be anything as it is plain English and can be more than one word), and choose `new` in the **On Click** menu. Click **Apply**.



Name button1
Label Spiral
On click New...
Visible [✓] Frozen [ ]

Drag the **corner** of the button to resize it, and drag it by its **centre** to relocate it to the lower part of the page.



This creates a new procedure such as this one in the Procedures Pane.

```
to button1_click
; This is a comment to help you remember the purpose
of this procedure. Use an instruction like this when
the button is clicked:
; FORWARD 100
end
```

The grey lines, between the title line that starts with `to` and the `end` line, are just comments in plain English - read and delete the grey lines (but keep the `title line` and the `end line`). Instead, type the instructions to be executed when you click on the turtle.

For example:

```
to button1_click
setcolour random 140
spiral 0 123
end
```

Give it a try! **Clean** the page, drag the turtle around and click on the button. Do that again. If there is one spiral you don't particularly like, click on the **Undo** button immediately after the action.

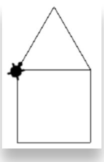
When you are done with this project, save it - the small red dot indicates that there is something to save. When you leave your project, always remember to save it first.

Then click on this icon to go back to **My Projects**.

More advanced ideas

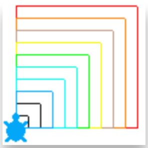
Super and sub procedure challenge:

Can you figure out a super procedure `house` that uses the sub-procedures `square` and `triangle` to create this?

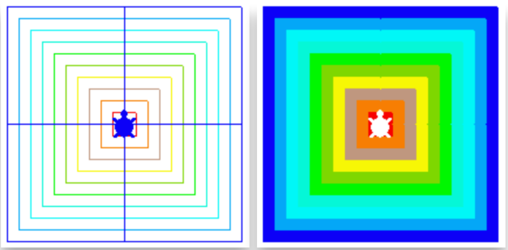


Pattern challenge:

You made this pattern earlier in this project.



Can you make four times that, deployed like a fan? Filled in too?



Spiral challenge:

Here's a gift - with a catch. It is yours if you can figure out what all these variables do. The changes are highlighted.

```
to newspiral :size :angle :limit :increase
if :size > :limit [stop]
forward :size
right :angle
newspiral :size + :increase :angle :limit :increase
end
```

The good news is that this procedure can do any spiral you can think of. The bad news is that it now requires **FOUR** inputs. Try these, use `cg` as needed:

```
newspiral 0 125 200 2
newspiral 0 90 100 1
newspiral 0 90 200 10
newspiral 0 125 200 2
```

Good to know

Note that the **recursive** line, the last line where the procedure calls itself, must have the **same** number of inputs as found on the title line. Four in this example. `:size + :increase` are added together and constitute the first input.

Curriculum Links for Ontario

C3.1 - Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves sequential, concurrent, repeating and nested events.

C3.2 - Read and alter existing code, including code that involves sequential, concurrent, repeating, and nested events, and describe how changes to the code affect the outcomes.

Arts D1 - Creating and Presenting: apply the creative process to produce a variety of two and three-dimensional art works, using elements, principles and techniques of visual arts to communicate feelings, ideas and understandings.