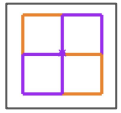
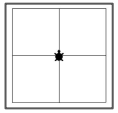
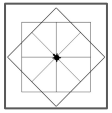
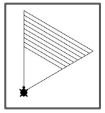
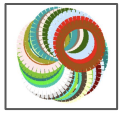
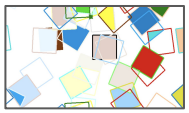
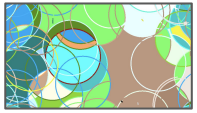
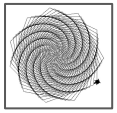





Geometric Fun

with

LYNX™

lynxcoding.org

With funding from

Canada

CODEtoLEARN.CA

TakingITGlobal
INSPIRE INFORM INVOLVE
INSPIRER INFORMER ENGAGER

Geometric Fun!

V2.1

2 Introduction!

Check out the **Getting Started Guide** before you try these cards. Or...**jump right in!**

View it in the **User Guides** under the Help Menu at **lynxcoding.org**

Use these cards to explore some simple geometric and coding ideas!

You will learn about:

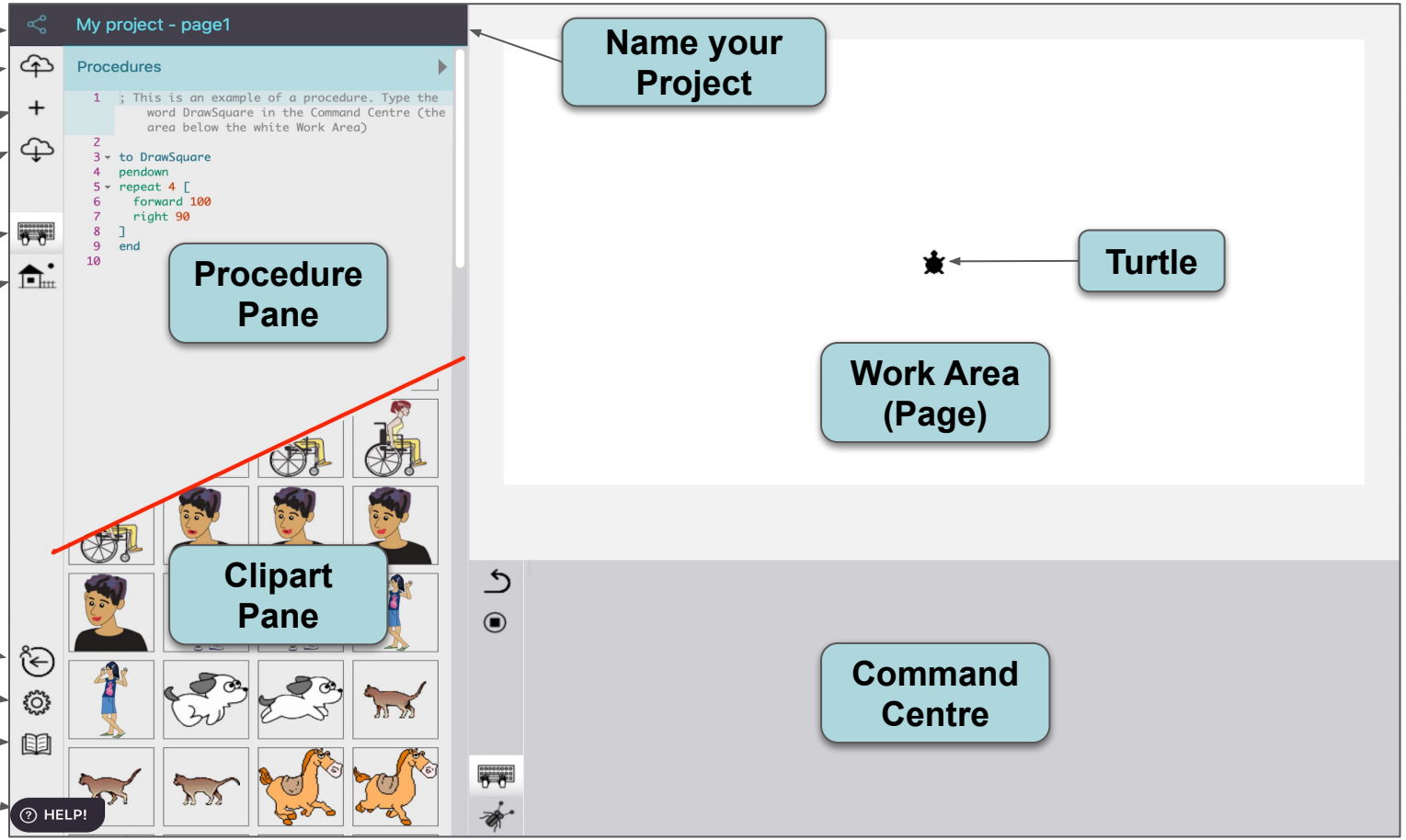
- The Total Turtle Trip Theorem
- Polygons
- Subprocedures & superprocedures
- Random
- Variables
- Conditionals

- Consider doing these cards in order as they build on one another.
- Be sure to share ideas, struggles, and challenges regularly with your friends.
- Challenge your friends by making up challenges for them.
- Post those in your classroom—real or virtual!
- **IMPORTANT: Name your project and SAVE often!**

Geometric Fun!

V2.1

3 Layout of Lynx



Geometric Fun! 4 Start Drawing

pd (pendown)
pu (penup)
pe (penerase)

fd (forward) **bk** (back)
rt (right) **lt** (left)

wait (wait)
e.g., **fd 100 wait 2 rt 50**

cg *(**C**lears **G**raphics and puts the turtle in the centre of the page)*

In the Command Centre, type:

pd Press **Return**
fd 100 Press **Return**

pd puts the pen down. The number is an *input*. **fd 100** is a command telling the turtle to move forward 100 turtle steps (pixels).

Try these:

rt 60 *(turn right 60 degrees)*
bk 150 *(move back 150 steps)*
lt 145 *(turn left 145 degrees)*

Type many commands on one line, then press Return. 'Arrow up' to the same line and press Return again!

Experiment with putting the turtle's **penup** and **pendown** before you use **forward** or **back** commands.

pu (**p**en **u**p)
pd (**p**en **d**own)

Can you:

- Make a dotted or dashed line.
- Print your name.
- Find out how many pixels (turtle steps) wide or high the work area is?
- PLAY! Try small and big numbers! Try numbers less than 1. :-)

GEOMETRIC FUN! 5 REPEAT + [ ]

Repeat is a command that saves you lots of work!

Square Brackets [ ]
Square brackets are used to contain a set of instructions—in this case to be **'repeated'.**

You will find the square brackets on your keyboard up near the Return key.

Type the following in the Command Centre and press Return:

pd
repeat 10 [fd 100 bk 90 rt 3]
This means, "Do this ten times: go forward 10, go back 90, turn right 3."

Try these. (Use **cg** to clear graphics—if desired!)

repeat 20 [fd 100 rt 165]
repeat 8 [fd 70 bk 60 rt 45]
repeat 10 [fd 100 rt 140 bk 100 rt 45]
repeat 6 [fd 80 rt 60 bk 80 lt 120 wait 2]
repeat 20 [fd 80 rt 18 wait 2 bk 80 fd 10 wait 2]

*LOOK! A **repeat** inside a **repeat!***

repeat 10 [repeat 15 [fd 4 rt 15] rt 120]
repeat 9 [repeat 10 [fd 4 rt 20] rt 120]

Add **wait** commands to slow some of them down a bit.

Try changing one input at a time in some of those examples.

Try putting **pu** and **pd** in various spots inside some of those examples.

Try doing the same line over and over again. Pretty cool, eh! Can you write a **repeat** command to do that automatically?

Geometric Fun! 6 Make Procedures

You write procedures in the Procedure Pane.

You will have many procedures. Make sure each procedure ends with the word **end** on a separate line!

If you don't, you'll get this error message:

I don't know how to 'procedurename'

A procedure is a group of LYNX instructions with a name you choose. It becomes a new command. Note: it only works inside the project you are working on now!

PARTS OF A PROCEDURE
A procedure has three parts:

to wiggle	*title line:* **to** — then a space, then a single word for the name of the procedure
fd 80 rt 90	*body:* instructions for the turtle and other objects like text boxes and buttons
end	*last line:* this must ONLY be the word **end**

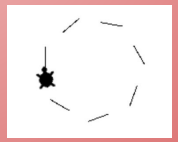
Click in the Procedure Pane.

Type something like:
to wiggle
fd 80 rt 60 bk 80 lt 120
end

Type **wiggle** in the Command Centre.

Change the inputs (values) in the procedure. Try it again.

Can you write a procedure to make this?



Total Turtle Trip

When a turtle takes a trip and ends up with the same heading, it has completed a Total Turtle Trip — 360 degrees.

In this case, there are **4** turns. Each turn is **90** degrees.

4 X 90 = 360


```
to square
repeat 4 [forward 100 right 90 wait 2]
end
```

Larger or smaller?

If you want to make a larger or smaller square, which number would you change?

Yes. the **forward** command.

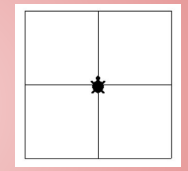
If you want to draw the square on the other side, what command do you need to change?

Yes. Change **right** to **left.** Or use **back** instead of **forward.**

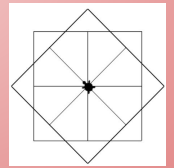
Make the smallest square you can.
Make the largest square you can.

Can you make this pattern?

*(Hint: Make a square. Change the **fd** or **rt**. And so on.)*



Or this one…
*(Hint: Make a square, then add a command in Command Centre, then make **square** again.)*



Make other patterns that you dream up! :-)

Geometric Fun!

V2.1

8 Colour and Size

setc or **setcolour**
setbg set background

setc “red makes the turtle **red**

setbg 49 makes the background **olive**
You don’t need the “

There are 140 colours 0 - 139!

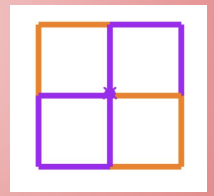
setpensize 10 will make a thicker line

Minimum size is 1.
Maximum size is 30.


```
to bluesquare
setc “blue
setpensize 8
repeat 4 [forward 100 right 90 wait 2]
;setc “blue makes the square blue
end

to squareright
setc "violet
setpensize 5
repeat 4 [fd 100 rt 90 wait 2]
end

to squareleft
setc "orange
setpensize 5
repeat 4 [fd 100 lt 90 wait 2]
end
```

Modify the following **wiggle** procedure by adding **setpensize** and **setc**

```
to wiggle
fd 80 rt 60 bk 80 lt 120
end
```



Can you make this pattern?

Can you change the background colour using **setbg**?

Experiment!

Total Turtle Trip (again!)

When a turtle takes a trip and ends up with the same heading, it has completed a Total Turtle Trip — 360 degrees.

An equilateral triangle is a triangle that has 3 sides of equal length.

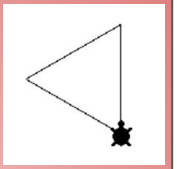
Making Triangles!

So, you now know the Total Turtle Trip Theorem! You know that the number of the **repeats** times the **angle** = 360!

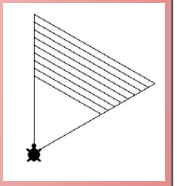
Think about the procedure below - what inputs would you add to create an equilateral triangle? Ready to try it?

```
to triangle
repeat  ? [ fd 100 rt ? ]
end
```

Can you make an equilateral triangle that goes to the left? Or an equilateral triangle with shorter (or longer) sides?.



Can you make a pattern similar to this.



Make a triangle pattern of your choice.

Think about using **pu, pd, setc, setpensize**

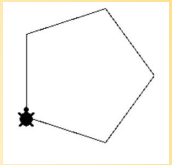
The word **polygon** comes from two Greek words.

The word **'poly'** means 'many' and **'gon'** means **'angle'**.

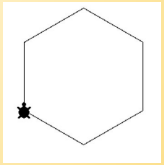
The word **'polygon'** means '**many angles**'.

Let's Make Some Polygons!

to pentagon
repeat 5 [fd 100 rt ??]
end

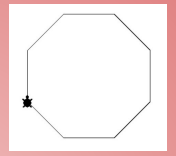


to hexagon
repeat ?? [fd 100 rt 60]
end



What is the **rt** angle for each of these? Remember the Total Turtle Trip!

Make an octagon.



Make a decagon.

Make a polygon with as many sides as you can!

Make a polygon but have the computer do the arithmetic for you to figure out the angle!

Note: You may have to make your sides shorter!!

Don't forget about the Total Turtle Trip!

You may have discovered the circle as you played with **Other Polygons**!

Don't forget about **setc** and **setpensize**.

Think about the **circle** procedure below - what inputs would you add to create an circle? Ready to try it?

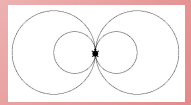
to circle
;remember the Total Turtle Trip
repeat ?? [fd ?? rt ??]
end

What is a circle after all?

Once you discover it—you can have all sorts of fun!

Can you make:

- a very small circle.
- a very large circle.
- the circles go in different directions!
- some 'googly eyes'!
- a snowman?



Create other shapes!
Share them with your friends!

You can often make the same pattern in different ways!

Be careful with the next examples.

Watch out for those decimal points!

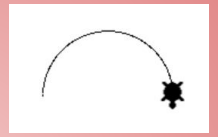
Try these.

```
to circle1
repeat 360 [fd 1 rt 1]
end

to circle2
repeat 180 [fd .5 rt 2]
end

to circle3
repeat 720 [fd 1 rt .5]
end

to circle4
repeat 720 [fd .5 rt .5]
end
```

Why do **circle1** and **circle4** look the same? Ask your friends to explain it!

Make the same size circle several different ways!

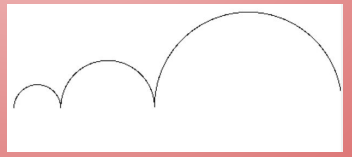
Make a **semicircle**.



Make several different size semicircles.

Try to make a pattern like the one below.

Hint:
You'll have to turn your turtle between semicircles. Or find another solution! :-)



Geometric Fun!

V2.1

13

Subprocedures!

A **subprocedure** is a procedure that is used inside another procedure.

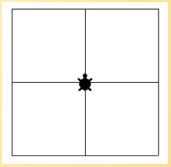
A **superprocedure** is a procedure that contains a **subprocedure**.

This allows you to make complex programs out of simpler bits!

Remember this challenge from the Simple Squares card?

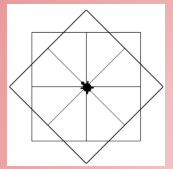
It is much easier with a superprocedure and a subprocedure!

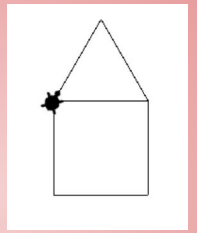
```
to square
pd
repeat 4 [fd 100 rt 90 wait 2]
pu
end

to 4square
repeat 4 [square rt 90]
end
```



square is now a **subprocedure** inside **4square.**

Write a **procedure** to make these patterns.





Compare your solutions with your friends!

Is one solution more efficient than another? Why?

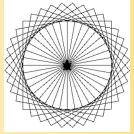
GEOMETRIC FUN! 14 SQUARES – REPEATED!

Square is a subprocedure inside **squaremore1**

Total Turtle Trip
When a turtle takes a trip and ends up with the same heading, it has completed a Total Turtle Trip — **360 degrees**.

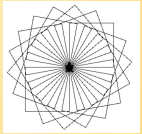

```
to square
pd
repeat 4 [fd 100 rt 90 wait 1]
pu
end

to squaremore1
repeat 36 [square rt 10 wait 2 ]
end
```

36 X 10 = 360



```
to squaremore2
repeat 18 [square rt 20 wait 2 ]
end
```

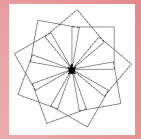
18 X 20 = 360



Can you modify the **squaremore3** procedure to create this picture?

```
to squaremore3
repeat 9 [square rt ?? wait 2 ]
end
```

9 X ?? = 360



What would the angle be if you change the **repeats** as follows? Try them!

- Change the **repeat** to **12.**
- Change the **repeat** to **360.**
- Change the **repeat** to **180.**
- Experiment with other repeats and angles!

Can you make the computer calculate the angle to turn?

Geometric Fun! 15 Colour + Size

setc = setcolour

setc random 140 picks a random colour between 0 and 139 — which is 140 colours!

setpensize 1 + random 30 picks a random pensize between 1 and 30.

***Note**: you need the '1 + random 30' because pensize cannot be zero*

Random is useful in games, simulations, math, etc.

Experiment with **Colour** and **Pensize!** And **Random!**

Let's use our **squaremore1** procedure. Add in a random colour change.

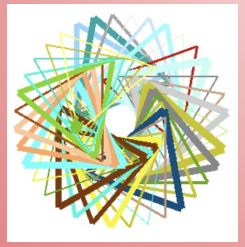
```
to squaremore1
repeat  36 [square rt 10 wait 2 setc random 140 ]
end
```

Then add in a random pensize too!

```
to squaremore1
repeat  36 [square rt 10 wait 2 setc random 100  setpensize 1 + random 30 ]
end
```

Use some of your other polygon procedures to make colourful patterns!

Share your artwork with your friends!



Geometric Fun!

V2.1

16

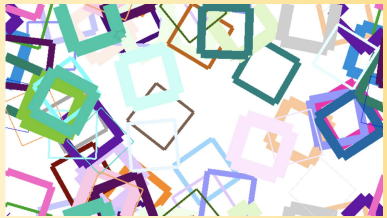
Colour + Size (Again!)

Decomposition
It is good practice to create small procedures (mind-sized bites)!

Breaking down a complex problem into smaller, more manageable parts is **decomposition**.

It is a cornerstone of **computational thinking.**


```
to square
repeat 4 [fd 100 rt 90 wait 2]
end

to move
pu
rt random 360
fd random 300
pd
end

to squaredance
repeat 50 [move setc random 140
setpensize 1 + random 30 square]
end
```



Move is a subprocedure inside **squaredance**.

Make some changes to the **move** procedure.

- Change the **rt random** number.
- Change the **fd random** number.

Make similar changes to the **squaredance** procedure.

Advice: Just make one change at a time so you can see the effects.

Geometric Fun!

V2.1

17

Fun with Fill!

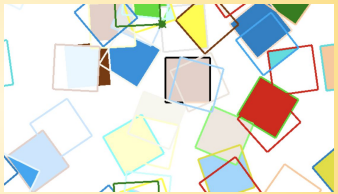
fill = fill with colour!

setpensize 4 makes a solid square so the **fill** won't leak!

forever Do what's inside the brackets until stopped.

To stop, click the 


```
to square
repeat 4 [fd 100 rt 90 wait 2]
end

to paint
pu rt 20 fd 20 pd
setc random 140
fill
end

to move
pu rt random 360
fd random 300 pd
end

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

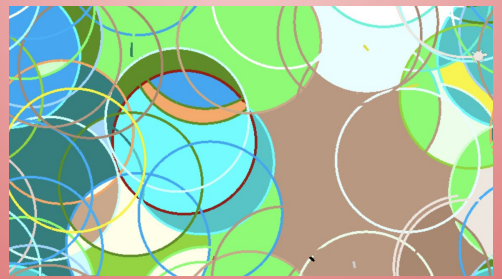


In **move**, change:

- the **rt random** number
- the **fd random** number.

Make similar changes to the **paint** procedure.

Try this with other polygons!



In mathematics, a variable is a quantity that can change. Letters are used to represent these changing, unknown quantities. In Lynx, we can use words.

:size is called a local variable.

:size is a choice of words. It can be any word, like **:length**

Any word works as long as you have a colon (:) before it.


```
to square :size
pd
repeat 4 [fd :size rt 90]
end
```

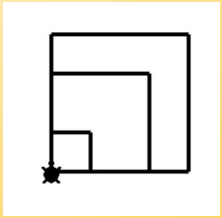
The variable **:size** must be in the title line and in one place in the body of the procedure.

Now try these in the Command Centre:

square 40

square 100

square 140



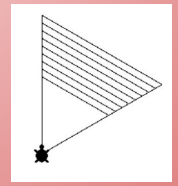
Make a **circle** procedure with a local variable.

Make other polygon procedures with local variables.

Do you remember this challenge from the Triangle card?

It's easier with **variables!**

Try it!



Here we are using **two** local variables:

:step for line length
:angle for the angle

NOTE: The **spiral** procedure is 'calling itself'. It has itself inside as a subprocedure!

But, it adds **2** to **:step** each time!

spiral :step + 2 :angle


```
to spiral :step :angle
setpensize 1 ;to reset size to 1
if :step = 100 [stop]
;This conditional statement says if :step equals 100, the procedure stops. This prevents the turtle from spiraling forever.
forward :step
right :angle
spiral :step + 2 :angle
end
```

Type: **spiral 2 90**

Figure 1 spiral 2 90

Type: **spiral 2 60**

Can you figure out why there is a difference?

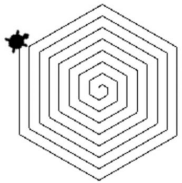


Figure 2 spiral 2 60

Let's play with the numbers a bit.

Change **+ 2** in **spiral :step + 2 :angle** to other numbers.

Change it back to: **spiral :step + 2 :angle**
Try **spiral 3 60**

OH OH!!!! What happened?!

Hint: the conditional is:
if :step = 100 [stop]

Hmmmmm… is "**= 100**" a problem?

What other math symbol might be better?

Geometric Fun! 20 More Spirals!

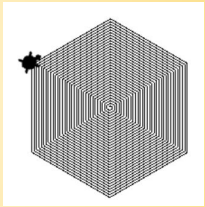
NOTE:
Now we are using **>** instead of **=**

if :step > 100 [stop]

Also, we changed the size of the increase in **:step** to **.5**

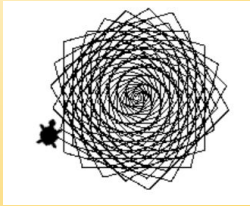
spiral :step + .5 :angle

to spiral :step :angle
if :step > 100 [stop]
;This conditional statement says if :step **is greater than** 100, the procedure stops. This prevents the turtle from spiraling forever.
forward :step
right :angle
spiral :step + .5 :angle
end



Type: **spiral 1 60**

Type: **spiral 2 75**



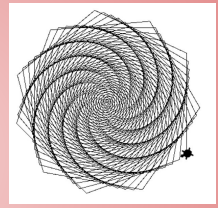
Can you explain the difference?

Change **if :step > 100 [stop]** to:
if :step > 200 [stop]

Type:

spiral 2 65

spiral 2 170



OK!!

Put **setc random 140** inside your spiral procedure.

Go for it!! Experiment!

Share!

Reflect on Your Learning!

You have learned:

- **pu, pd, fd, bk, rt, lt**
- **cg, wait, repeat, [ ]**
- **setc, setpensize, fill**
- **setbg, random**
- **forever, stop**
- **to, end,** procedures & subprocedures
- Total Turtle Trip
- conditionals **if**
- variables

Make Art

Although there is much more to learn, you are well prepared to experiment with what you know!

You can make wonderful art now.

Revisit all the cards to help you think about how you might make some art.

Your art might be 'static' — a lovely picture.

Or, your art might be 'dynamic' — with **wait** commands so you can enjoy the creation!

You might want to create a **setup** procedure — similar to this one. Then write a procedure such as **art**.

```
to setup
setc random 140
pu
rt random 360
fd random 200
pd
end

to art
setup
;your lines of commands and
subprocedures go here!
end
```