



**TakingITGlobal**  
INSPIRE INFORM INVOLVE  
INSPIRER INFORMER ENGAGER



With funding from

**Canada**

**L NX**

**Projects for  
Learning to Code**



**For kids 9+**

© Logo Computer Systems Inc. (LCSI), 2020. All rights reserved.

No part of the document contained herein may be reproduced, stored in retrieval systems or transmitted, in any form or by any means, photocopying, electronic, mechanical, recording or otherwise, without the prior approval from Logo Computer Systems Inc.

ISBN 978-1-7774107-0-4

Printed March 2022 and printed in Canada.

Principal contributors: Michael Quinn, Peter Skillen, Alain Tougas  
Additional contributions by: Elena Yakovleva, Brenda Sherry, Marina Klimova.



**LCSI**® is a registered trademark and **L\*NX** is a trademark of Logo Computer Systems Inc.



**TakingITGlobal** is a registered trademark of TakingITGlobal.

# Table of contents

<b>Introduction.....</b>	<b>2</b>
<b>Welcome to LYNX .....</b>	<b>4</b>
<b>Project 1 - Geometric fun .....</b>	<b>12</b>
<b>Project 2 - The art of the LYNX .....</b>	<b>22</b>
<b>Project 3 - Interactive thank you card .....</b>	<b>34</b>
<b>Project 4 - Secret codes .....</b>	<b>54</b>
<b>Project 5 - Kill the virus.....</b>	<b>68</b>
<b>Project 6 - Pong!.....</b>	<b>82</b>
<b>Appendix A. Top 50 LYNX primitives.....</b>	<b>100</b>
<b>Appendix B. Common coding errors .....</b>	<b>112</b>
<b>Appendix C. LYNX colour chart.....</b>	<b>116</b>
<b>Appendix D. Finding clipart, sounds, and screen captures.....</b>	<b>118</b>
<b>Appendix E. FAQ.....</b>	<b>120</b>
<b>Appendix F. Sharing.....</b>	<b>122</b>
<b>Appendix G. Musical Notes .....</b>	<b>127</b>

# Welcome to LYNX

## About CanCODE 2021-2024

CanCode is a Federal Government program administered by ISED Canada. The objectives of CanCode are to provide digital skills training to K-12 students and teachers in Canada and to promote awareness and interest in coding and robotics.

The long-term goal is to make Canada a leading innovation economy with a diverse and inclusive workforce.

CanCode funding is directly responsible for the availability of this book as the writers, editors, artists, and printers have received Government of Canada funds for their efforts.

## What an expert says

*We have successfully integrated LYNX into our culturally responsive mathematics project in a number of Grade 3 to Grade 8 classrooms in Ontario. For this project, we explored the interaction between LYNX coding and Indigenous design, technology and artistry as part of our exploration of the mathematics of Algonquin loom beading and Métis finger weaving.*

*Using LYNX, students analyzed the structure of the beadwork or weaving, articulated that structure by writing code, and then wrote new procedures to create new beading or weaving designs.*

*The dynamic nature of LYNX allowed students to explore the different interpretations of mathematical concepts inherent in the work, which then allowed us to more fully analyze students' mathematical understanding.*

*The students loved writing code for their turtles to create beadwork or weaving designs. They created more and more sophisticated procedures to progress from, for example, creating a single bead, to using the repeat command to create multiple beads, and ultimately full bracelet designs.*

*The designs that the turtles created, using the code written by the students, did not always align with the students' intended creations. This provided the students numerous, wonderful opportunities for problem solving!*

Ruth Beatty, PhD  
Faculty of Education  
Lakehead University





# Welcome to LYNX

## Get ready for this coding adventure

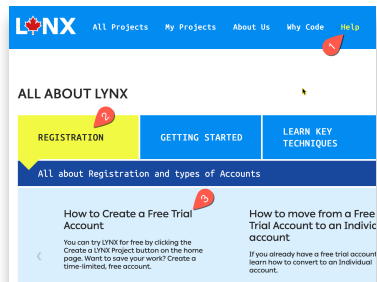
In many ways, coding is good for you. It forces you to think and solve problems. Knowing how to code will probably help you with school projects right now but it will certainly give you the 21<sup>st</sup> century skills you will need to succeed in the next 15 years.

If you are ready to move on from block coding but not really enticed by these “grown-up picky-syntax” languages like Javascript, LYNX offers a text-based programming environment, both serious and fun at the same time. The skills you learn with LYNX will ease your transition to JavaScript or Python. Plus, it makes project management and sharing super easy, with cloud and social media integration.

Better yet, LYNX was designed by Canadians and includes Canadian spelling and Canadian clipart and themes.

## Create an account!

To code the projects in this book you need to create a time-limited **Trial Account**. Look for extensive instructions in the **Help** section of the LYNX web site. Go to [lynxcoding.club](http://lynxcoding.club), click on **Help**, then **Registration**. Read the document **How to Create a Free Trial Account**.



There is also a video covering this topic in the same section of the Help page.

If you are enjoying coding with LYNX and since Canadians can use LYNX for **free** we suggest you upgrade to a permanent **Individual** account. In the same Help section, look for the document **How to move from a free trial account to an individual account**.

After you have registered, every time you start LYNX, you need to Login. Look for these words in the top right corner. Once you have logged-in, your login name will appear there.

Login/Register

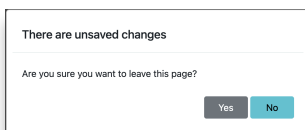
## Saving

**Important:** There is **NO Auto Save**. You must save your work regularly by clicking the **Save** icon.



You may see a small **red dot** under the cloud. This indicates that your project contains something that has not been saved recently. This is a reminder to save! You don't want to lose your work!

Whenever you try to leave the LYNX Editor (closing a window, going to My Projects, etc.), LYNX *may* display this dialog box:



This is your browser saying: this page has some unsaved work. If you leave this page now, *you will lose your latest work*. Stay on that page and use the **Save** icon before leaving if you wish to save the last changes you made. If you just saved, this dialog box won't show.

## Help and resources

Besides the book you are looking at right now, LYNX offers plenty of help...

### INSIDE THE LYNX EDITOR WHILE WORKING ON YOUR PROJECT

#### TOOLTIPS

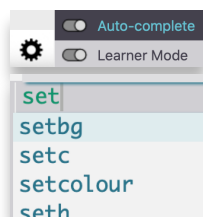
Leave the mouse pointer on a LYNX **primitive** (built-in vocabulary) and you will see something like this image on the right: a short definition and example. This is especially useful to find out the type of input some primitives require (in this example, **setcolour** needs a number). If you like, copy the example and paste it into the Command Centre or in a procedure.

```
SETCOLOUR colour-name-or-number
Sets the colour of the turtle's pen.
Often used with setpensize, pd, fill, forward,
back.

setcolour 54
setcqlour
```

#### AUTO-COMPLETE

If you turn on **Auto-Complete** in the **Settings** menu, which we suggest for beginners, as you type in the Command Centre or in the Procedures Pane, LYNX will display a list of primitives that match what you seem to be typing. Click on the line you want.

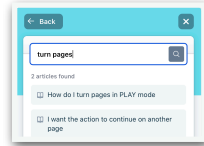


# Welcome to LYNX

## HELP WIDGET

HELP!

The **Help widget**, located at the bottom-left of the window, opens a floating box where you can type questions and get related explanations. Type very short questions or concept words.



## ONLINE LYNX DICTIONARY



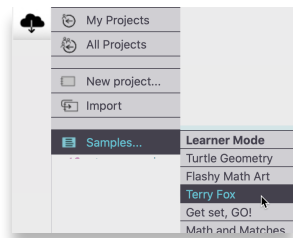
The small book icon in the bottom-left corner of the window opens a floating window containing a good dictionary of LYNX primitives, organized by topics (graphics, text...). The explanations and examples are more complete than those of the tool tips. It also contains the chart of LYNX colour names and numbers.

## LYNX BUILT-IN SAMPLES



Open a sample from the list in the **Down from the cloud** menu. You can play with it, look at the code and comments or copy some code for your own project.

Be aware though, in order to see a sample project, you must leave the project you are currently working on. If need be, take the time to save your project before you go see a sample project.



## LYNX COLOUR-CODED FORMATTING

In the Procedures Pane and in the Command Centre, your instructions are automatically colour-coded by LYNX as you type:

```
1 to square :size
2 ; Turtle and text box required
3 repeat 4 [forward :size right 90]
4 print sentence 'This square is: ' :size
5 end
```

LYNX primitives (built-in vocabulary) are **green**. Procedure names (additional vocabulary added by you in the project) are **teal**. Inputs (values required for primitives and procedures to work), including variable names, are **red**. Comments are **grey**. Plain text is **blue**.

The **grey** comments are simple explanations of what the code is doing. They are not code! Comments help other people understand what your code is doing.

## IF YOU ARE OUTSIDE THE LYNX EDITOR

### PDF DOCUMENTS AND VIDEOS

Visit the LYNX home page ([lynxcoding.club](http://lynxcoding.club)), click on **Help**. You will find lots of PDF files and videos with plenty of detailed explanations about registrations and accounts, short easy-to-follow activity cards, longer projects, a primitive cheat sheet, the colour chart, and good resource materials like a 30-page **Getting Started** book, the complete list of (200!) LYNX primitives, detailed explanations about the grammar and syntax of the LYNX language, and how to organize your projects (folders, public, private, properties, deletion, etc.) and share projects.

### LYNX TEMPLATES AND SAMPLE PROJECTS IN ALL PROJECTS PAGE

All Projects

At [lynxcoding.club](http://lynxcoding.club), go to the **All Projects** page, and open the **Templates** folder. These projects contain some code, and often, clipart that you may want to use for your own project. If there is a template you like, simply open it, **change its name**, and **save** it; it will then be your own project, in your private LYNX cloud area (you will now see it on your **My Projects** page).

The same applies to the sample projects in the other folders. The folder **Learner Mode** contains simple projects, and the folder **Advanced**, well, more advanced projects. There are other themed folders like Math, Games, etc.

### CANCODETOLEARN.CA

This comprehensive web site [cancode.to/learn.ca](http://cancode.to/learn.ca) ([sites.google.com/view/lynxcoding-org](http://sites.google.com/view/lynxcoding-org)) contains instructions for making projects, in the form of topic-based cards, project guides, short how-to videos and comprehensive webinar recordings, about 50 minutes each. Projects include a greeting card, probability problems, a hand-made calculator, a working ecosystem, a simulation of gravity, etc. You can view the materials online or download and print them all (PDF format).

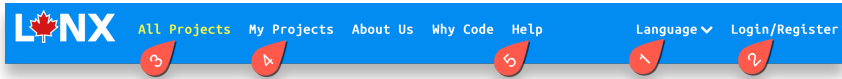
Take a look at our videos here: <https://www.youtube.com/playlist?list=PLA8nVBr85nK-LcnZ1bMsCp4fnuKmCxuqz>

# Welcome to LYNX

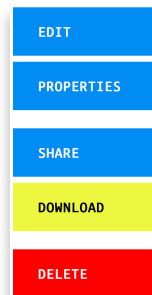
## The LYNX environment

### THE BASICS OF PROJECT MANAGEMENT

Let's tour the Home page ([lynxcoding.club](http://lynxcoding.club))



1. Our website appears in **English**. In the top right corner, you can choose French, or a few Indigenous languages, in the **Language** menu.
2. Once you're logged in, you are on the LYNX Home page, you will see your name in the top, right corner instead of "**Login/Register**".
3. **All Projects** is where you will find all the projects that other LYNX users have made public, plus some projects, samples and Templates prepared by Team LYNX.
4. **My Projects** is your private area where you will find your own projects. Click on a project to see it in **Player mode** (no Command Centre, no Procedures or Clipart Pane - just the project as others will see it), then choose one of these options to **Edit** the project, change its **Properties** (title, preview image etc), **Share** the project with friends, **Download** a copy to your computer (backup), or **Delete** it (careful, you **can't** recover a deleted project).
5. **Help** is where you will learn everything you need to know about LYNX:

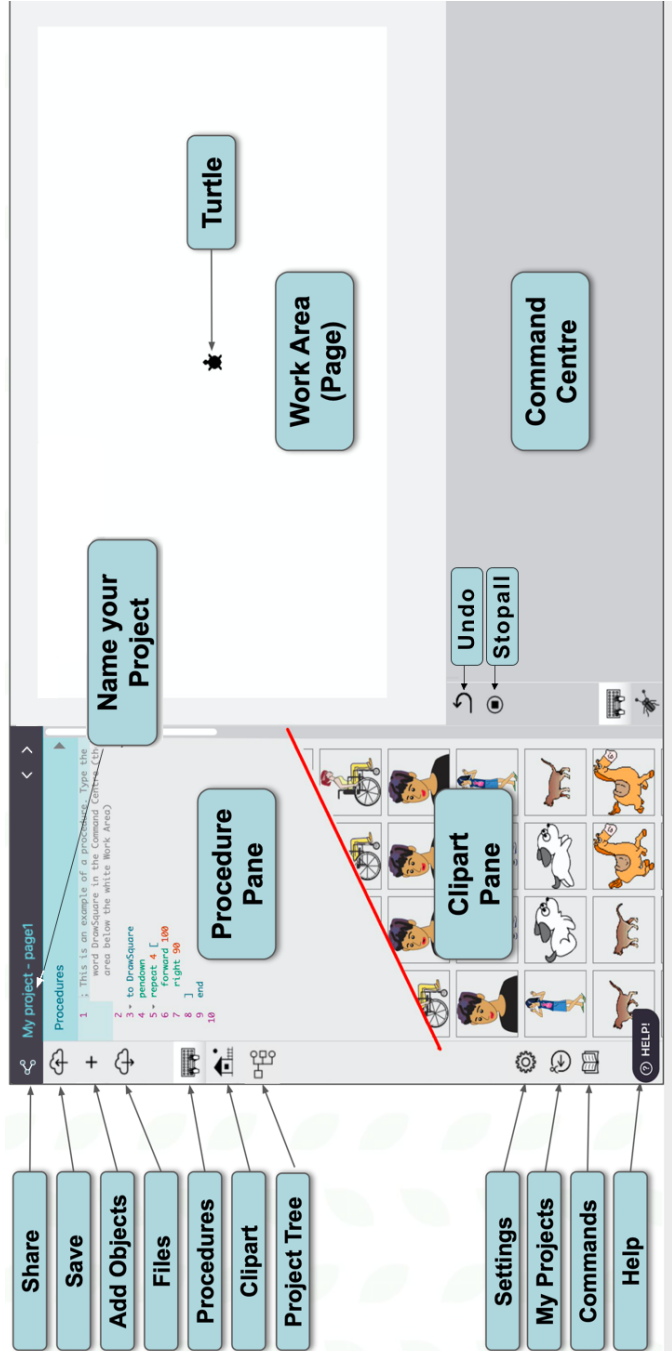


- All about LYNX: Registration. Getting Started, Learn Key Techniques, Sharing a Project, Quick Code Videos.
- Projects & Lessons: Secret Path Project, Terry Fox Project, Activity Cards, Theme Based Videos, Theme Based projects.
- Resource Materials: Cheat Sheet, Colour Chart, List of LYNX Primitives, Creating a School Account, Vocabulary & Syntax, Organizing your projects, How to Share a Project, Exploring Computer Science with LYNX & this book in PDF format.



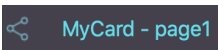




Take the time to explore at least the titles and descriptions of these resources. For now, **Login**, and on your **My Projects** page, click on **Create a LYNX Project** (big yellow button).

## THE LYNX PROJECT EDITOR











These are the components of the LYNX editor. Read more about them on the next pages or in the Help section.



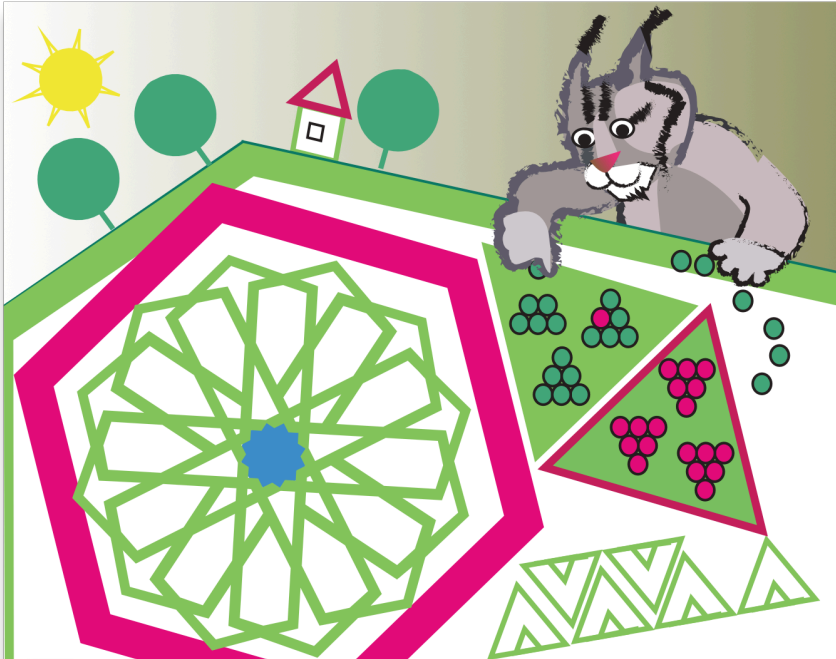
# Welcome to LYNX

<p>Work area (page)</p> 	<p>Your project happens here. Think of it as a blank canvas or empty stage. A project can have multiple pages.</p>
<p>Command Centre</p> 	<p>Below the Work area you type instructions here that are executed <i>immediately</i>. This is a good place to “practice” things. Think of it as your rough draft area. This area is NOT part of your project, therefore it is not saved with your project.</p>
<p>Name your project</p> 	<p>Click on this field to give your project a new name. If you don't, <i>all</i> your projects will be called “My Project” - not a good idea!</p> <p>This field also displays the name of the <i>current page</i>, and arrows to turn pages.</p>
<p>Procedures icon and Procedures Pane</p> 	<p>Click on the <b>Procedures</b> icon to display the Procedures Pane. This is where you create procedures. More about procedures later.</p>
<p>Clipart icon and Clipart Pane</p> 	<p>Click on the <b>Clipart</b> icon to display the Clipart Pane. This is where you store clipart that you can use as backgrounds and turtle shapes.</p>
<p>Project tree</p> 	<p>Click on this icon to see all the pages and all the objects found in each page. Click on the small triangle to expand a page, click on an object to select it, then click on <b>Edit</b> to open its dialog box, or on <b>Delete</b> to delete the page or object. If you delete something, it is gone!</p>
<p>Share</p> 	<p>Click on this icon to choose ways to share your project with friends.</p>



Save		This icon saves your project in its current state. <b>Save often as there is NO autosave.</b> If you see a <b>red</b> dot below the cloud it means you have not saved in a little while. It is a reminder to <b>Save Now</b> .
Add objects		This icon opens a menu with commands to create new turtles, buttons, text boxes, etc.
Files		This icon opens a menu with commands to go to My Projects, all public projects, create a new project, or open some sample projects.
Settings		Use these options to use LYNX in Dark or Light mode, choose the font size of primitives, and work the way you like.
My projects		Quick access to your projects. Don't forget to save your project before you leave it!
Commands		This icon opens a new window with plenty of information about LYNX primitives and its colour palette.
Help		Another good place to get help. Enter a question here (ex: "how to animate a turtle") and see the results.
Undo		Click on the <b>Undo</b> icon immediately after running an instruction that produced an undesirable result.
Stopall		Click on the <b>Stopall</b> icon to stop everything that is happening in a project. If the action started by clicking on a button or a turtle, you can click on that button or turtle again to stop the action.
Debug		This debugging tool will help you as a programmer. You can set a breakpoint (to pause your program) at any line of your code and see what is happening on the screen and also the values of all your variables at this point.

# Project 1 - Geometric fun



This project will introduce you to LYNX. You will learn about the Command Centre and the Work area (or page), a few LYNX primitives / commands, some geometry concepts like angles, polygons, the total turtle trip concept (the big 360), and the basic foundation of coding: creating your first procedure!

## First things first, start a new project

Go to the LYNX web site ([lynxcoding.club](http://lynxcoding.club)) and log into your account (If you don't know how to do that or don't have an account, see **Create an account** on page 4). Make sure you are on **My projects** then, click on

CREATE  
A LYNX PROJECT



As a good habit, start by **naming your project** in the **top left corner**. Change **My project** to a name that will make your project recognizable from the list of projects on your **My Projects** page. If you don't give it a name, all your projects will be called **My project** and you will have a hard time finding out which is which!




Second good habit: save your project often. There is NO autosave. Simply click on this icon:



If you have to leave the project before it is completed, just remember to save before you leave. You will find your new project on the page **My Projects** when you are ready to continue.

## Your first instructions: turtle's pen, moving and turning

You're looking at a new, blank project, with a turtle in the centre of the work area (or the page).

No turtle on your page?  
Choose Turtle in the  menu!

In the grey Command Centre below the white Work Area, type:

**pendown**  
**forward 100**

**PD** IS THE SHORT FORM. PRESS ENTER  
**FD** IS THE SHORT FORM. PRESS ENTER

**Pendown** puts the pen down so the turtle leaves a visible line. **Fd 100** is a command telling the turtle to move forward 100 turtle steps (pixels). We say that **fd** is the **command**, and **100** is the **input**.

Try these:

**right 60**  
**back 150**  
**left 145**

TURN RIGHT 60 DEGREES. **RT** IS THE SHORT FORM.  
MOVE BACK 150 STEPS. **BK** IS THE SHORT FORM.  
TURN LEFT 145 DEGREES. **LT** IS THE SHORT FORM.

Type many commands on one line, then press **Enter**. Remember that primitives like **forward**, **back**, **right** and **left** always require an Input. You need to tell the turtle how far to move or how wide to turn! Don't forget to leave a space between the **command** and the **input**.

# Project 1 - Geometric fun

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



Make a dotted or dashed line.

Print your initials.

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. What does **forward -150** do?

Try a number larger than 9999, you will see your first **LYNX error message**. Error messages are important. Read them, they tell you how to fix the small problem you are having.



## Now that your turtle is moving...

..add some colour and different line sizes! Here are some graphics commands you can use:

<b>setcolour</b> (or <b>setc</b> for short)	SETS THE COLOUR OF A TURTLE AND PEN
<b>setpenseize</b>	SETS THE TURTLE'S PEN WIDTH
<b>setbg</b>	SETS THE COLOUR OF THE BACKGROUND IN THE WORK AREA
<b>cg</b>	STANDS FOR <b>C</b> LEAR <b>G</b> RAPHICS. THE TURTLE ALSO GOES HOME
<b>clean</b>	CLEANS THE PAGE, WITHOUT MOVING THE TURTLE
<b>home</b>	MOVES THE TURTLE BACK HOME, IN THE CENTRE OF THE PAGE

Try these instructions, change the input values, make your own combinations:

```
cg
pd setpenseize 20
fd 50
clean
setcolour 'blue'
rt 90
fd 30 pu fd 10 pd fd 30 pu fd 10 pd fd 30
```



### GOOD TO KNOW

Be careful with the single straight quotes around the colour names! You will find other colour names in Appendix C, at the end of this book.

You can change the background colour of the page too:

```
setbg 'yellow'
```

Unlike the turtle graphics, the background colour remains even after a `clean` or a `cg` command. You will have to use `setbg 'white'` to return to the original background colour.

### GOOD TO KNOW...

Did you notice that **SOME** commands work as “just one word”, while some other require a value?

`Cg, clean, penup, pendown` work “as is”- 1 word

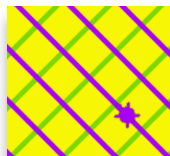
`Forward, back, right, left, setpensize` all want a number as an input. `Setcolour` and `setbg` want a colour name or number.

Explore these commands some more:

```
cg
pd
setpensize 5
rt 45
setcolour 'green'
fd 9999 fd 9999
lt 90
setcolour 'violet'
fd 9999 fd 9999
setpensize 1
```

9999 IS THE LARGEST NUMBER YOU CAN USE

BACK TO NORMAL PEN SIZE



# Project 1 - Geometric fun



Usually, you can undo the effect of a *few previous commands* by clicking on the **Undo** button immediately after the action.

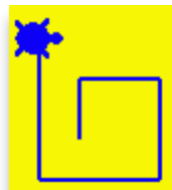


You can execute instructions many times without retyping: Type this instruction all on one line in the Command Centre:

```
cg
setc 'blue' forward 30 rt 90
```

YOU CAN TYPE SEVERAL  
INSTRUCTIONS ON A LINE

Press **Enter** and see the results. Now click the **Up Arrow** on your keyboard to return to the line with the LYNX code. Change **forward 30** to **forward 60** and press **Enter** again. Using Arrow keys you can re-execute any line already present in the Command Centre. Change **forward** again with **90** or **180**.



When you want a clean empty Command Centre, type:

```
cc
```

STANDS FOR **CLEAR COMMAND CENTRE**

## Can you repeat that?

Type the following in the Command Centre and press **Enter**:

```
cg
repeat 10 [fd 100 bk 80 rt 6]
```

**CG** STANDS FOR CLEAR GRAPHICS

In plain English, this means,

**Repeat ten times whatever instructions are inside the 2 square brackets. In this case, go forward 100, go back 80, turn right 6 degrees.** Find square brackets to the right of the letter P on the keyboard.

Try these (use **cg** before each instruction, if you wish):

```
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 down how fast your commands are executed. it will be easier to see each command running and then understand what is going on.

Something like:

```
repeat 8 [fd 70 wait 5 bk 60 wait 5 rt 45]
```

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?

## Your first procedure

A **procedure** is a group of LYNX instructions with a name you choose. The **procedure's name** becomes a **new command**, just like **forward** and **right**. Procedures that you create *only work inside the project you are working on now!*

Click on the **Procedures** icon to open the **Procedures Pane**.



Click in the Procedures Pane, where the lines are numbered, and type this procedure:

**A procedure has three parts**

Procedures

```
1 to wiggle
2 fd 80
3 rt 60
4 bk 80
5 lt 120
6 end
```

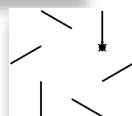
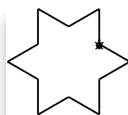
→ The title line: the word **to**, a space, and the **name** of the procedure as a single word (no space)

→ → → Instructions: As many as you want, one or many instructions per line.

→ The word **end**, alone on the last line

Type **wiggle** in the Command Centre. Try it several times actually. Make sure your Pen is down (**pd**). Change the inputs (values) in the procedure. Try it again.

Can you write a procedure to create this design by using **pu** and **pd**?



# Project 1 - Geometric fun

## Many squares, many styles

Create this procedure in the Procedures Pane.

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

Now type **square** in the Command Centre.



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

Yes. The input of the **forward** command.



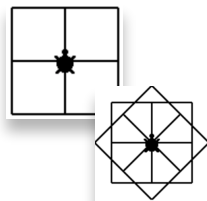
If you want to draw the square on the other side, what command do you need to change?

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 these patterns?

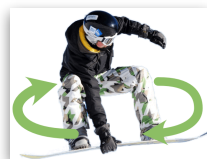


**Hint:** Make a square, make the turtle turn a bit (using a **rt** instruction in the Command Centre), then make another square, and so on.

Make other patterns that you dream up! :-)

## Total turtle trip, let's do triangles!

Look at your **square** procedure again. How much does the turtle turn in total? **Four times 90 degrees**, right? Do the math, **4 x 90 is 360 degrees**. Like doing a 360 on a skateboard or a snowboard. That is the **Total Turtle Trip**, or **TTT** for short!



The same is true for a triangle (or any polygon).

### GOOD TO KNOW

**In order to make a closed figure, the turtle must turn a total of 360 degrees.**



Think about the procedure below - what inputs would you use to create an equilateral triangle? Do the math and try!

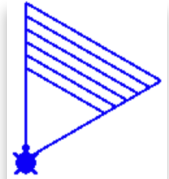
```
to triangle
repeat ?? [fd 100 rt ??]
end
```

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`, `setcolour`, `setpensize`

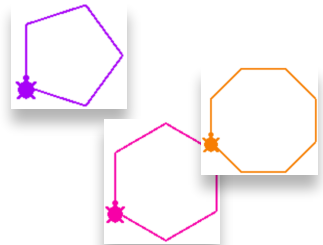


## Some polygons now!

Following the “**total turtle trip**” rule above, can you figure out how to make these other polygons?

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

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



What is the `rt` angle and the number of `repeat` for each of these?

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!

# Project 1 - Geometric fun

## And for the grand finale, circles!

Think about the **circle** procedure below - what inputs would you add to create a circle? Ready to try it? Remember the TTT!

```
to circle
setpensize 2
repeat ?? [fd ?? rt ??]
end
```

THICKER LINE, READ MORE ABOUT IT BELOW

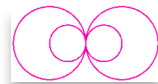
What is a circle after all? **Hint:** you may have to use a very small input for **forward**, otherwise your circle will go wayyyy over the edge of the page.

Once you discover the “trick” of the circle, you can have all sorts of fun!

Can you make:



- a very small circle? A very large circle?
- some “googly eyes” (circles to the right, circles to the left)?
- a snowman?



If you made googly eyes and want to fill the smaller inside eyes with yellow, drag the turtle inside one smaller circle and type this in the **Command Centre**: **setcolour 'yellow' fill**. **Setpensize 2** makes a thicker line so the colour doesn't leak through the edge of the circle.

Not sure what **fill** does? Remember to use Tooltips! Place your cursor over the word **fill** for 2 seconds. Now you can't see the turtle, because it is a **yellow** turtle inside a **yellow** eye. Run **setcolour 'black'** to see the turtle again, and drag it to the other eye to fill it too (use **setcolour 'yellow'** again before filling).

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

to circle2
repeat 180 [fd 0.5 rt 2]
end

to circle3
repeat 720 [fd 1 rt 0.5]
end
```

```
to circle4
repeat 720 [fd 0.5 rt 0.5]
end
```

## More advanced ideas

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

Can you make the same size circle several different ways?

Can you make semicircles? Of different sizes?

Try to make a pattern like this one.

Hint: You'll have to turn your turtle between semicircles.



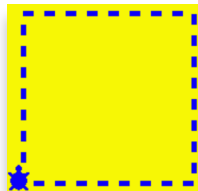
Can you make this using a **square** and a **triangle** procedure?



Can you make a **DashedLine** procedure to produce this?



Good! Now can you use this procedure to create a dashed square?



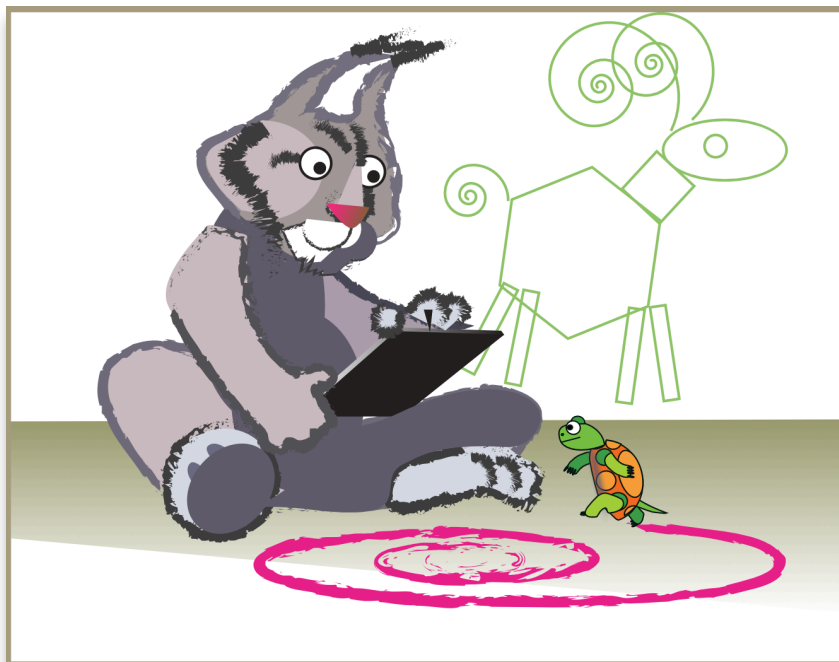
## Curriculum Links for Ontario

**MATH C3** - Solve problems and create computational representations of mathematical situations using coding concepts and skills.

**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.

## 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 Project**.



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!

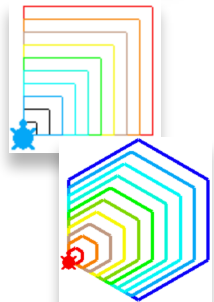
To create a procedure with an input, add the input on the title line, preceded by a colon (: **size**). It can be any name but must be a single word. No space between the colon (:) and the name (size)!

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?

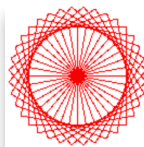
Can you create another polygon procedure with an input, to create polygons of any size?



## Project 2 - The art of the LYNX

### 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 many_squares :size
repeat 36 [square :size right 10]
end
```

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

Clear the graphics and type this in the Command Centre:

```
setcolour 15
many_squares 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                                     THIS MEANS CLEAR THE COMMAND CENTRE
repeat 100 [show random 80]
44
6
62
37...
```

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:

```
cg  
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
```

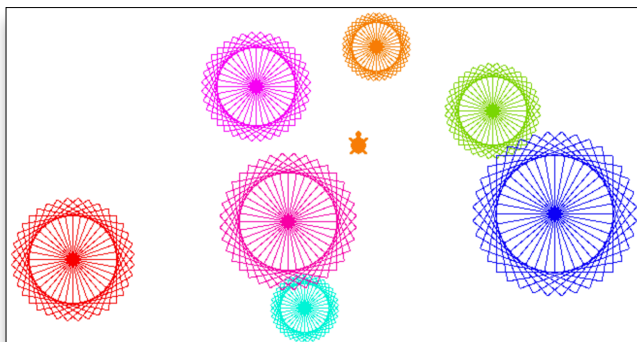
Yes, LYNX chooses a **random size** to be given to **forward**. Let's add **random colours** to the recipe.

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 [changelcolour square :size right 10]
end

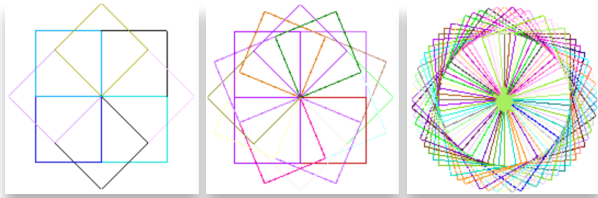
to square :size
  repeat 4 [forward :size right 90]
end

to changelcolour
  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 **changelcolour** to choose a random colour.



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

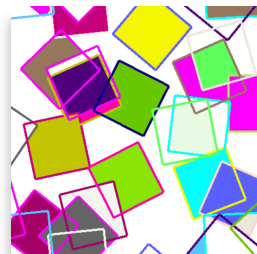


### I can go on forever like that

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
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.

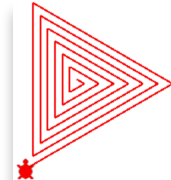


### Spirals - I have my conditions!

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` before you try and make sure the pen is down.

```
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.

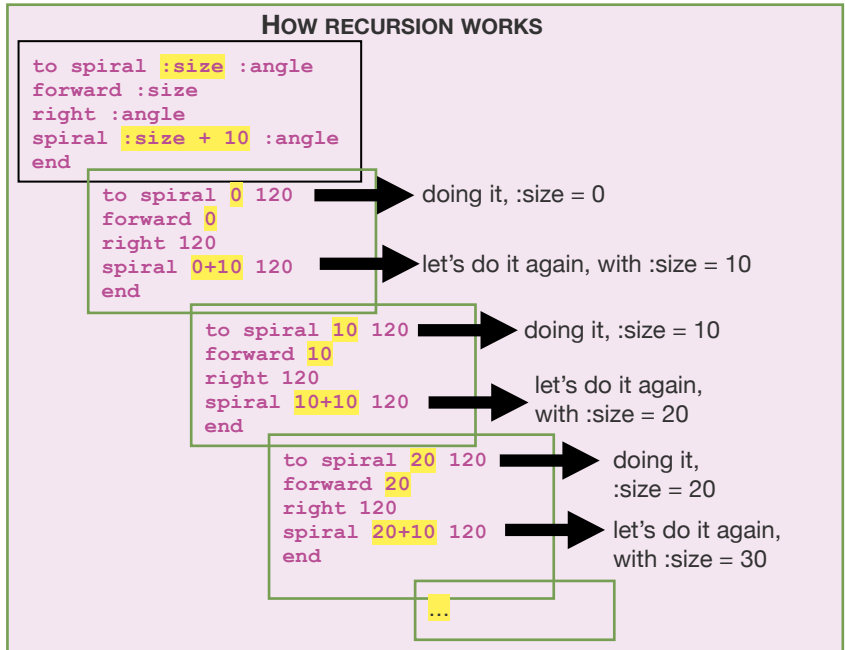
Type this in the Command Centre.

```
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):



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

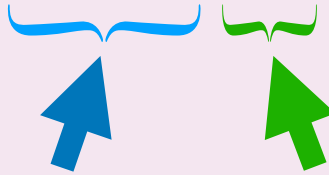
```
to spiral :size :angle
if :size > 200 [stop]
forward :size
right :angle
spiral :size + 10 :angle
end
```

The "if" instruction is a **CONDITIONAL STATEMENT**, in this case it is a **STOP RULE**. It stops a recursive procedure.

## Project 2 - The art of the LYNX

### HOW IF WORKS

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



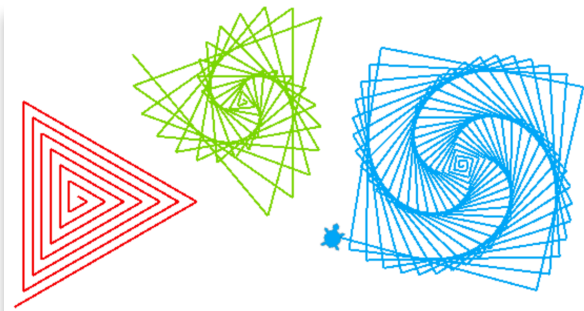
```
IF THIS-IS-TRUE THEN-DO-THIS
```

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)?

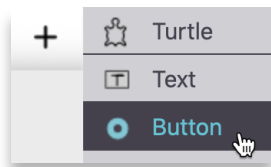
## Click me!

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**.

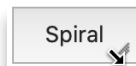
A button, named **nothing** by default, appears in the centre of the page. A button is just a visible object that runs code when clicked. Right-click on the button to open its dialog box.



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**.

A screenshot of the Button dialog box. It has fields for Name (button1), Label (Spiral), and On click (New...). There are checkboxes for Visible (checked) and Frozen (unchecked). At the bottom are buttons for Apply and Cancel.

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.

## Project 2 - The art of the LYNX

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.



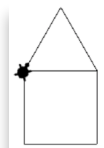
Then click on this icon to go back to **My Projects**.



### 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?

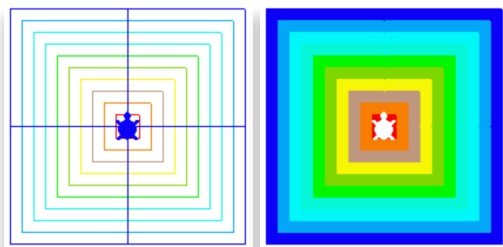


#### PATTERN CHALLENGE:

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.

## Project 3 - Interactive thank you card



Now for something new. Project 3 introduces **clipart** (turtle shapes and backgrounds), **animation**, **text boxes**, **multiple pages** and **page navigation**, **buttons** and **sounds**.

You have certainly bought or made a paper birthday card. This project shows you how to make a digital *Thank You* card to send to frontline or essential workers. Of course, the same skills can be used to make any other type of card: for a birthday or Mother's Day...Ready?

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

As a good habit, start by **naming your project**. And remember to save often.

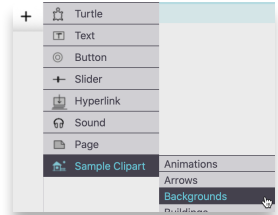




## Create a background scene

Basically, you need to put a shape or costume on the turtle. Click on the **+** menu, choose **Sample clipart**, then choose **Backgrounds** in the sub-menu.

The left pane will now become a **Clipart Pane** and will contain a set of backgrounds you can use.



In the Clipart Pane, each box is numbered. You can see # 1 in the image on the left.

Type this in the Command Centre:

**setshape 1**

USE ANY NUMBER IN A BOX THAT HAS CLIPART

If you wish, use **setsize** to make it bigger or smaller. In the Command Centre, type:

**setsize 55**

THE DEFAULT SIZE OF THE TURTLE IS 40

This looks good, but it is still just a turtle wearing a costume. You can drag your turtle / background around (try it). Use the command **home** if you want to put the turtle in the centre of the page. Now, type this to **stamp** the turtle on the page:

**stamp**

THE TURTLE COSTUME IS “PRINTED” ON THE BLANK PAGE

It may seem that nothing really happened, but the turtle's shape (the background clipart) is now **stamped** on the page. You are looking at a **huge turtle** standing on **an image of itself**. Try to move it around: the image that moves is the turtle, the one that doesn't is the background:



## Project 3 - Interactive thank you card

To eliminate any confusion between an active turtle and stamped background, set the turtle's shape back to its original shape:

**setshape 0**

THE DEFAULT NUMBER OF THE TURTLE IS 0



Now you have both an active turtle and a nice background (as shown above). If you want to erase the background, and perhaps other lines or drawings, use:

**cg**

OR THE LONG FORM, CLEARGRAPHICS

### A QUICK WAY TO GIVE A SHAPE TO A TURTLE: CLICK & CLICK

- Click on a clipart you like. The mouse pointer turns into a hand, holding the clipart.
- Now click on the turtle to give it that shape.
- Then, use stamp to make it a background image.



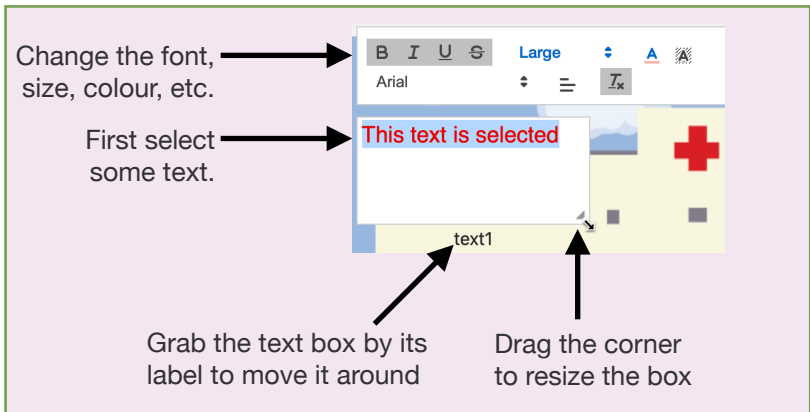
## Create a text box and write your message

Cards you buy in a store have words already in them and then usually you add some words of your own. Let's do that here.

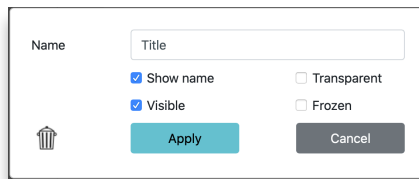
Click on the **+** menu then choose **Text**.

A text box appears on the page. **Move** it anywhere by dragging the Name (**Text1**) and **resize** it by using the arrow in the bottom right corner.

Type in the title of your "Thank You" card, select the text and **format the text** using the options provided (the next picture shows you how to move and resize the text box).



Right-click on the text box to open its dialog box. Change its name to **Title**. The name must be a **single word**, no space anywhere (not even at the end - remember that). Then click **Apply**.



### About the several options in the Text box dialog box

If ☐ **Show name** is unchecked, or if ☒ **Transparent** is checked, you can't move the text box around.

If ☒ **Frozen** is checked, you can't resize or move the text box, but you can still type in it. It is frozen in the place you put it.

If ☐ **Visible** is unchecked, oops, it's not deleted, but you can't see it anymore! There are two things you can do about this:

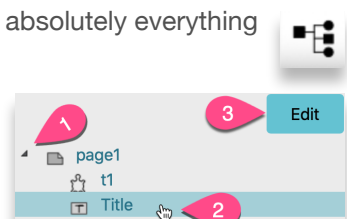
- Type this in the Command Centre (don't forget the comma):

**title, showtext**

**TITLE, MEANS "I AM TALKING TO YOU"**

or...

- Click on the **Project tree** icon to see absolutely everything your project contains.
  1. Click on the **triangle** besides **page1** to reveal its contents.
  2. Click on the **name** of the text box.
  3. Click on **Edit** to open its dialog box and check the box ☒ **Visible**.

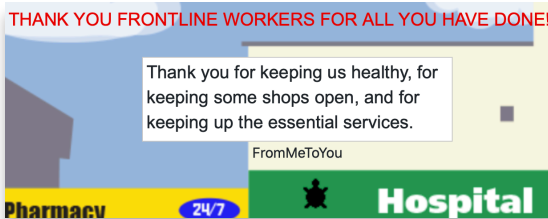


## Project 3 - Interactive thank you card

Make adjustments to the text so it looks nice on the page. This example shows a **transparent** text box. You will not see its label.



Add a new text box. Open its dialog box and name it **FromMeToYou** (one word, no space). Type your *Thank You* message in that text box, and format it to your liking. As an example, we wrote some words for you.



### Let's hear it!

Just for fun, type this in the Command Centre:

```
show frommetoyou
```

```
Thank you for keeping us healthy, for keeping...
```

You see, the **name of the text box** (**frommetoyou**) is also a command that reports the **contents of the box**.

Here is a nice way of using this feature. Type this in the Command Centre:

```
say 'hello'
```

THE COMPUTER READS ALOUD HELLO

```
say frommetoyou
```

THE COMPUTER READS THE CONTENTS OF THE BOX

You should **hear** the computer reading the message. If you don't, check the volume on your computer.

The command **frommetoyou** reports the contents of the text box as a big long word. So instead of just reading **'hello'**, LYNX reads the big long word.

You can use different voices for reading text. Type this in the Command Centre to get a list of all the voices available on your computer (may be different or not available at all on your computer):

`show voices`

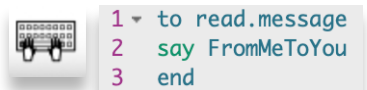
`Alex Alice Alva Amelie Anna Carmit Chantal...`

Your list will be different. Try other voices using `sayas` instead of `say`:

`sayas frommetoyou 'anna'`

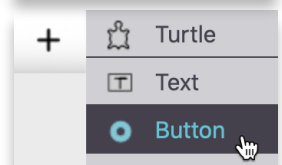
Keep in mind that this *Thank you* card is something to share. People other than you will **NOT** have a Command Centre to run this instruction. So let's create a **procedure**, and a **button** to do that.

First, create this **procedure** in the Procedures Pane:



```
1 to read.message
2 say FromMeToYou
3 end
```

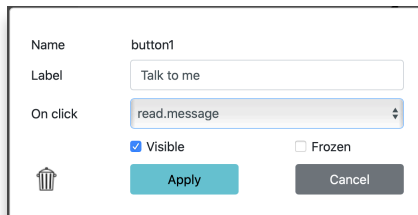
Then, click on the **+** menu and choose **Button**.




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.

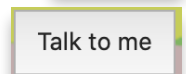
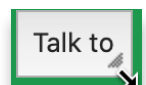
In the dialog box that appears, type **Talk to me** as the label (it can be anything as it is plain English), and choose **read.message** in the **On Click** menu. Click **Apply**.



Name	button1
Label	Talk to me
On click	read.message
<input checked="" type="checkbox"/> Visible	<input type="checkbox"/> Frozen
	
<input type="button" value="Apply"/> <input type="button" value="Cancel"/>	

Now click on the button. Did you hear your message?

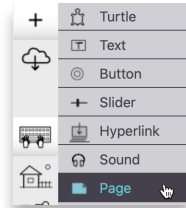
Drag the button by its centre to relocate it to the lower part of the page. Drag the bottom-right **corner** to resize it.



# Project 3 - Interactive thank you card

## Add a page

Just like cards bought in a store, your digital card should have more than just a cover page. Let's add a second (inside) page to make this card more interesting. Click on the **+** menu then choose **Page**.



BLANK! Don't worry, your work is safe. This is a **new blank page**. Look above the Procedures Pane. It says "page2" (a). Click on the **arrows** (b) to turn pages, but come back to **Page2** to continue making the card.



### GOOD TO KNOW

A page name (such as **page1** or **page2**) is also a **command** that goes to that page. That's why a page name must be a single word, no space, unlike commands like *forward* or *right* which require a space before the number.

You can rename or delete a page from the Project tree. Click on the page name and click on **Edit** to open its dialog box.

## Buttons to turn pages

There are a few ways to go from page to page.

1. The **arrows** above the **Procedures Pane** (as described above).
2. Typing the **name of the page** in the Command Centre.
3. Writing a **procedure** and using it in a **button** or in a **clickable turtle**.

Let's use the **button** method. Create this procedure:

```
5 to go.page1
6 page1
7 end
```

Then create a new button, open its dialog box, type **Go to page 1** (or anything you like; as explained earlier, this is plain English not code) as the label, and choose **go.page1** (the procedure) in the **On Click** menu.

Again, resize and relocate the button in the corner of the page (usually, you place a button that goes “back” in the bottom left corner.

Test the button. It should bring you to **Page1**. Now that you’re good at it, while you are on **Page1**, create a new button that goes to **Page2** (this requires a new procedure and a new button).

Instead of using a **Button**, add a **Turtle** to your Page. Right click on it and in the **On click** field, select **go .page1**. Move the turtle to the bottom, left corner. Click on the **+** then **Sample clipart**, then **Arrows, Buttons**. Click on a button image you like in the Clipart pane then click on the **Turtle** on the page. Test your clickable turtle.

## Your first animation: a moving ambulance

### NEW TURTLE, NEW BACKGROUND

Make sure that you are now on **Page2** (blank). There was no turtle to start with on **Page2**. Click on the **+** menu, choose **Turtle**.



Give it the shape of a city street scene:

**setshape 2** USE THE NUMBER OF A BACKGROUND CLIPART THAT YOU LIKE

Now use the same steps as you did before to create a background:

- Use the command **home** to centre the turtle on the page.
- Use the command **setsize number** to change the size of the turtle, if you want.
- Use the command **stamp** to stamp it on the page.
- Use the instruction **setshape 0** to see the turtle again.

### MORE CLIPART

You will need new clipart for the animation. Click on the **+** menu, choose **Sample clipart**, then choose **Nature, Other** in the sub-menu. This adds more clipart to your collection - you will have to scroll down to see them all; there is an ambulance towards the end.



### GET THE AMBULANCE TO MOVE

Drag the turtle (still has the default turtle shape) towards the bottom left side of the page, on the street. A turtle will always move in the direction its head is pointing. Make it point to the right - type this in the Command Centre:

**setheading 90** THIS MEANS “FACE EAST” - 90 DEGREES ON A COMPASS

## Project 3 - Interactive thank you card

It is a good idea to point the turtle in the right direction while it has a turtle shape - it is the only shape that shows its current heading.

When the turtle is “all good”, you can give it the shape you want.

Now that the turtle is “ready to roll”, give it the shape of the ambulance. Type this in the Command Centre:

```
setshape 34
```

USE YOUR NUMBER WHICH MAY BE DIFFERENT



There will be more than one turtle on that page. Giving them good names is a great idea. Right-click on the turtle (ambulance) and type **EMS** (Emergency Medical Services) as its name:

Name	<input type="text" value="EMS"/>		
Xcor	<input type="text" value="-279"/>	Ycor	<input type="text" value="-175"/>

Notice its **xcor** and **ycor** position also (-279 and -175 in this example, your values will be different). These are the X and Y coordinates of the turtle.

Now that the turtle has a name, you may as well use it when you talk to it. Type this in the Command Centre (don't forget the comma):

```
ems,
```

THIS MEANS: TURTLE NAMED **EMS**, I AM TALKING TO YOU

```
repeat 100 [forward 3 wait 1]
```

Cool, you have just added animation! Bring the turtle back to its starting position: type this in the Command Centre (use the **xcor** and **ycor** you saw in your turtle's dialog box above - you can round the values if you wish):

```
setpos [-280 -175]
```

SETPOS MEANS SET POSITION



## PUT THAT IN A PROCEDURE

If you want to launch the action from a clickable turtle or a button, you first have to make it into a procedure. Create one like this in the Procedures Pane.

```
13 ▾ to go.ems
14   ems,
15   setshape 34
16   setheading 90
17   setpos [-280 -175]
18   repeat 100 [forward 3 wait 1]
19   end
```

## RUN THE PROCEDURE FROM A BUTTON

Remember, when the card is shared with someone, it will not have a Command Centre to run its features. Create a new button, relocate and resize it like you did before.

Right click on the button to open its dialog box, and choose the procedure (**go . ems**) in the **On Click** menu:

Name	button4
Label	<input type="text" value="Click me!"/>
On click	<input type="text" value="go.ems"/>

Great! Close the dialog box and **click on the button** to launch the action. It should go to the starting position and ride for a few seconds.

## A clickable turtle: add a beating heart

You can easily do this by having a turtle grow and shrink. Stay on **Page2** and Click on the **+** menu, choose **Turtle**.

Right-click on the turtle to open its dialog box, name the turtle **Heart**.

Open the Clipart Pane and look for a **heart** shape near the end. Click on it, and your mouse pointer turns into a **hand**.



Now click on the turtle to give it that clipart as a shape. This is the same as running the instruction **setshape 38** (or whatever your number is).

Woah! This is a huge heart! Run an instruction such as **setsize 10**, or a number, higher or lower, that looks good on your page.

## Project 3 - Interactive thank you card

And for the animation, type this in the Command Centre:

```
heart, forever [setsize 20 wait 4 setsize 10 wait 4]
```

This tells your turtle (named **heart**) to run the instructions inside the square brackets *forever*. Basically, keep changing the size of the heart and wait a bit between each change. Click on **Stopall** in the Command Centre if you get dizzy!



If you want to launch the action just by clicking on the **heart**, you first have to make it into a procedure. Create one like this in the Procedures Pane. Choose different values for **setsize** if you like.

```
21 to heart.beat
22   heart,
23   forever [setsize 20 wait 4
24             setsize 10 wait 4]
25 end
```

Right-click on the Heart and select this procedure as the **On Click** instruction by using the Up and Down arrows in the **On Click** field:

Name	<input type="text" value="Heart"/>		
Xcor	<input type="text" value="0"/>	Ycor	<input type="text" value="0"/>
On click	<input type="text" value="heart.beat"/>		

### GOOD TO KNOW

Notice that you can click on the heart to start the animation, and click on it again to stop it. If you want the heart to change size only a few times. replace **forever** with **repeat 10**.

**Suggestion:** It may be useful to add a Text box on the page with a simple message like: *Click on the Heart*.

## Run Spot Run!

So far, we have things you could never have on a printed card. Let's add more! While on **Page2**, click on the **+** menu, choose **Turtle**.

Right-click on the turtle to open its dialog box, name the turtle **Dog**.

Click on the **+** menu, choose **Sample clipart**, then choose **Animations** in the sub-menu. Open the Clipart Pane and look for **dog** shapes. The dog shapes are pointing to the right. So set the heading of the new turtle-dog:



```
dog, setheading 90
```



Here's a new trick. If you give **more than one shape** to the turtle, it will **switch shapes** each time it moves (**forward, back**). Type this in the Command Centre:

```
setshape [69 70]
```

USE YOUR OWN NUMBERS WITHIN THE [ ]

The turtle is now wearing two shapes. Check this out: type this in the Command Centre:

```
forward 5  
forward 5  
forward 5
```

Drag the **dog** somewhere on the street, perhaps in front of the ambulance. Now open its dialog box to find out its position on the page. In this example, the position is **-110 -185**

Name	<input type="text" value="Dog"/>		
Xcor	<input type="text" value="-110"/>	Ycor	<input type="text" value="-185"/>

Now create this procedure to get the dog ready and then to run:

```
to dog.run
```

```
dog,
```

DOG, I'M TALKING TO YOU

```
setshape [69 70]
```

```
setheading 90
```

FACE EAST AND RUN LEFT TO RIGHT

```
setpos [-110 -185]
```

THE POSITION YOU SAW IN THE DIALOG BOX

```
repeat 100 [forward 4 wait 1]
```

CHOOSE NUMBERS YOU LIKE

```
end
```

## Project 3 - Interactive thank you card

### RUN THE PROCEDURE FROM A BUTTON

Create a new button, relocate and resize it like you did before.

Right click on the button to open its dialog box, and choose the action procedure (**dog.run**) in the **On Click** menu:

Name	button5
Label	Me too (arf arf)
On click	dog.run

Great! Click **Apply** to close the dialog box, and click on the button to try it. Wonderful!



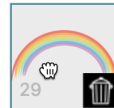
### Somewhere over the Flashing Rainbow

Here is another type of animation. No movement or shape switching. The turtle-rainbow will simply appear and disappear (hide, wait, show, wait).

Stay on **Page2** and Click on the **+** menu, choose **Turtle**.

Right-click on the turtle to open its dialog box, name the turtle **Rainbow**.

Open the Clipart Pane and look for a **rainbow** shape. Click on it, and your mouse pointer turns into a **hand**.



Now click on the turtle to give it that clipart as a shape. This is the same as running the instruction **setshape 29** (or whatever your number is).

Use a **setsize** instruction if you want to change its size. The default size is 40, so anything above 40 will be larger, below 40 is smaller.

For the animation, type this in the Command Centre:

```
rainbow, repeat 10 [ht wait 4 st wait 4]
```

**ht** and **st** mean **HideTurtle** and **ShowTurtle**. These instructions mean: do this 10 times: hide the turtle-rainbow, wait 4/10 of a second and show the turtle-rainbow, wait 4/10 of a second. **wait 10** means pause 1 second.

In the Procedure pane, make a procedure like this:

```
35 to flash.rainbow
36   rainbow,
37 repeat 10 [ht wait 4
38             st wait 4]
39 end
```

## RUN THE PROCEDURE FROM A BUTTON

Create a new button, relocate and resize it like you did before.

Right click on the button to open its dialog box. Type a label for the button and choose the action procedure (**flash.rainbow**) in the **OnClick** menu:

Name	button6
Label	<input type="text" value="Rain + sun = ..."/>
On click	<input type="text" value="flash.rainbow"/>

Finally, click on this button to try the animation. It will flash 10 times.

## Now make your words flash!

Now here's not one, but two magic tricks. Create this procedure in the Procedures Pane:


```
43 to startup
44   page1
45   title,
46 repeat 10 [hidetext wait 5
47             showtext wait 5]
47 end
```

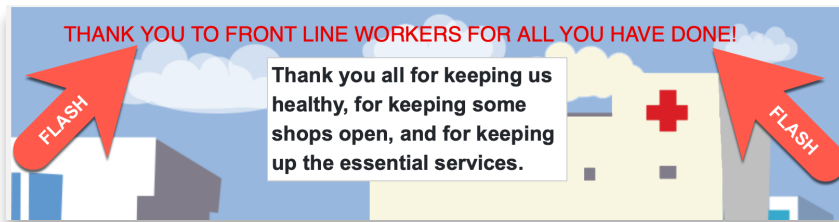
It is VERY IMPORTANT that you name this procedure **startup**, with this exact spelling. **Startup** is a special word. When someone opens your project, this procedure is executed automatically.

What does it do?

## Project 3 - Interactive thank you card

- **page1** is the name of the first page, and it is a command that brings up that page. It is important to do so when the project opens.
- **title** is the name of the first text box on that page. Add a comma (**title**,) to say “**title**, I’m talking to you”.
- **hidetext** means hide the Text box and **showtext** means show the Text box. The **repeat** instruction does this 10 times: make the text box invisible, then a delay of 5/10 of a second, then make the text box visible and add a delay of 5/10 of a second.

To test this, save your project (  ) and go to **My Projects** (  ) where all your projects are saved. Click on your **Thank you** project to see it in Play mode. You will be seeing Page1, and the **title** should flash.



### Let's make some noise: Add sound

The final touch: add a cheering sound for the opening of the project, when the title flashes.

#### FIRST, IMPORT A SOUND TO YOUR COMPUTER

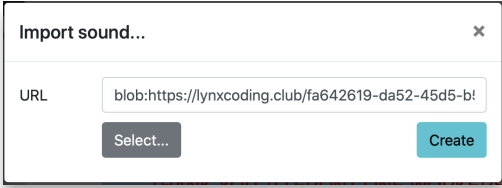
You can import WAV and MP3 sound files. If you don't have a cheering sound on your computer, you may go to this web site to find one:

[bbcsfx.acropolis.org.uk](http://bbcsfx.acropolis.org.uk)

Type **cheers** in the **Search** box and click on **Duration** to see the shortest sounds at the top of the list. In this example, we are using a short 10 second applause and cheers sound. Try the sound on the Web page, and if you like it, download it to your computer (probably in the **Downloads** folder).

## BRING THE SOUND INTO YOUR PROJECT

Back to your project. On **page1**, click on the **+** menu and choose **Sound**. This brings up the Import sound dialog box:

A dialog box titled "Import sound..." with a close button (X) in the top right corner. It contains a label "URL" followed by a text input field containing the URL "blob:https://lynxcoding.club/fa642619-da52-45d5-b!". Below the input field are two buttons: "Select..." and "Create".

Click on **Select** and locate the sound (WAV or MP3) file that you have downloaded to your computer. When you have it, the URL field will show the link. Now click on **Create**.

This creates a sound icon named **sound1** on your page. Click on it to hear the sound.

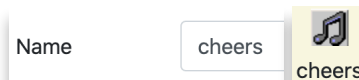


## PLAY THE SOUND

The sound has a name (**sound1**) and that name is also a command to play the sound. Type this in the Command Centre:

**sound1**

Great! Give it a better name now. Right-click on the sound icon to open its dialog box, and name it **cheers** (one word, no space).

A dialog box for renaming a sound icon. It has a label "Name" and a text input field containing the word "cheers". To the right of the input field is a small icon of a musical note inside a square frame, with the text "cheers" written below it.

## INCLUDE THE SOUND IN THE STARTUP PROCEDURE

Finally, include it in the **startup** procedure, so it plays when the project opens:

```
to startup
page1
launch [cheers]
title,
repeat 10 [hidetext wait 5
          showtext wait 5]
end
```

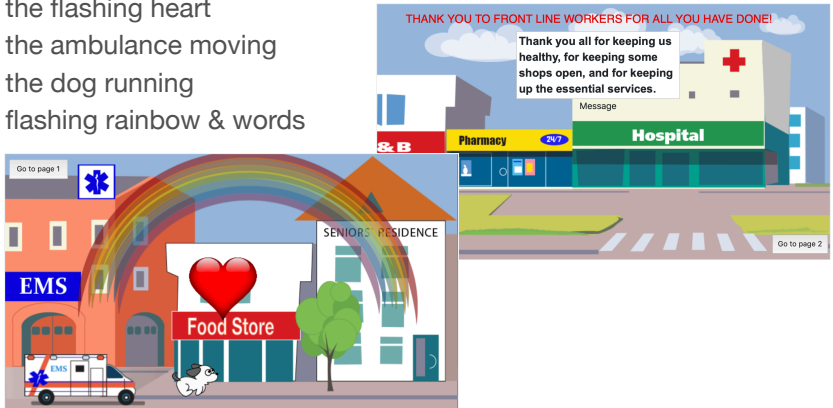
Why **launch**? **Launch** will start (launch) the sound and let it play on its own, and while the sound is playing, LYNX will immediately jump to the next instruction and flash the **title** text box. The cheering and the flashing will occur at the same time.

# Project 3 - Interactive thank you card

## Final test and debugging

You're a coder now... And good coders sometimes have... good bugs in their code! Time to test all the features of your project. Check for things that don't work as expected, and check for error messages in the Command Centre. Test:

- the button Talk to me
- the button that goes from page1 to page2
- the button that goes from page2 to page1
- the flashing heart
- the ambulance moving
- the dog running
- flashing rainbow & words



If everything is fine, you may right-click on the **sound** icon and uncheck the ☐ **Visible** box to hide it. It will still play the sound!

## Sharing the card

Nice work! Time to share it with some people, or with the entire LYNX community. *Appendix F* has all the instructions for sharing projects.

## All the procedures of this project

Just in case you need to check something, here are all the procedures in the project. Yours can be different, of course!

```
to read.message
say FromMeToYou
end
```

```
to go.page1
page1
end
```



```
to go.page2
page2
end

to go.ems
ems,
setshape 34
setheading 90
setpos [-280 -175]
repeat 100 [forward 3 wait 1]
end

to heart.beat
heart,
forever [setsize 20 wait 4
        setsize 10 wait 4]
end

to dog.run
dog,
setshape [69 70]
setheading 90
setpos [-110 -185]
repeat 100 [forward 4 wait 1]
end

to flash.rainbow
rainbow,
repeat 10 [ht wait 4
          st wait 4]
end

to startup
page1
launch [cheers]
title,
repeat 10 [hidetext wait 5
          showtext wait 5]
end
```

# Project 3 - Interactive thank you card


## More advanced ideas

### ADD YOUR OWN CLIPART

Add your own clipart to one of the scenes. It could be the logo of your local hospital or pharmacy, a wheelchair symbol, or anything related to your *Thank You* card.



Look in *Appendix D* for clipart resources on the Web, and special instructions regarding clipart with transparent surrounding. In short:

- Find a PNG image with a transparent surrounding. Often seen on a grey or a checkered background.
- Download the image to your computer or your work space in the cloud. Chromebook users, see Appendix E, FAQ.
- Open the Clipart Pane, select an empty spot and click anywhere on it to get this icon: . Click on it.
- This opens the Import image dialog box. Click on SELECT to find your downloaded image. Once the URL field is full, click on CREATE to add the new clipart to your Clipart Pane.

A screenshot of the 'Import image...' dialog box. It has a title bar with a close button (X). Inside, there is a label 'URL' next to a text input field containing the word 'URL'. Below the input field are two buttons: 'Select...' and 'Create'.

Create a new turtle and use the new clipart as the turtle shape, and set the size of the turtle, not too big, not too small. You can always adjust this detail later.

```
setshape 4  
setsize 10
```

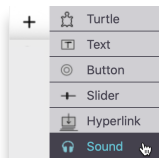
USE YOUR OWN CLIPART NUMBER  
TRY DIFFERENT NUMBERS

### AUTOMATIC PAGE TURNING

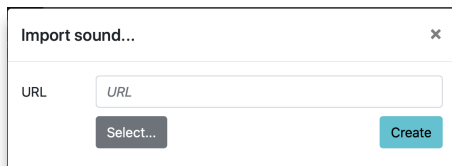
Something else you can do: look at your **startup** procedure. You can, after the Text Box flashing, add an instruction that goes automatically to **page2**. Remember, a page name is also an instruction that goes to that page.

## RECORD YOUR VOICE

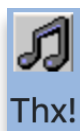
If you know how to record your voice on your computer, you can record a **SHORT** message and save it as a **WAV** or **MP3** file on your computer. *Appendix D* suggests some voice recording tools. Then...



Choose **Sound** in the **+** menu. This opens the Import sound dialog box.



Click on **Select** to navigate to the sound file you have recorded and saved, then click on **Create** to create the sound icon on the page.



Click on the icon to hear your recording.

If you wish to rename the sound icon, right-click on the icon to open its dialog box, and type a new name. Remember: use a one-word name, no space.

You can move the sound icon anywhere and let the user click on it to hear your message, or you can:

- Create a procedure that plays the sound, and run this procedure from a button  
**to hear.me**  
**thx!**  
**end**

USE THE NAME OF YOUR SOUND ICON

- Include this command (**thx!**) inside an existing procedure.
- Include this command in the **startup** procedure, so it happens automatically.

## CURRICULUM LINKS FOR ONTARIO

- **C3.1** - Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves conditional statements and other control structures.
- **C3.2** - Read and alter existing code, including code that involves conditional statements and other control structures, and describe how changes to the code affect the outcomes.

## Project 4 - Secret codes (i\*l\*o\*v\*e\*)



Let's do something totally different now. No turtles or graphics (unless you want to add some), but lots of fancy text manipulation. Secret codes have been used for ages, even Caesar, two thousand years ago, had his own “recipe” for encrypting messages.

You should know that you can talk to and give commands to the **cursor** (the flashing bar inside the text box): move the cursor around and have it insert, delete, or change the text as it moves. Let's try this!

The cypher (encoding) in this project is pretty simple, nothing like spies and governments use nowadays, but it is a fun starting point. The process, for you, will resemble this:

- Create a text box for your message.
- Think (pseudo-code) about a way to “scramble” your text.
- Create a procedure that manipulates the text, in a way that matches your pseudo-code.
- Create a procedure that does the exact opposite, to decode the scrambled text.

## A plain background

Start a new project and, as a good habit, start by **naming your project**. Remember also to **save** your project often!



You are going to create two encryption methods on two separate pages. Put a nice background on Page1. Type this in the Command Centre:

```
setbg 'violet'
```

SETBG STANDS FOR SET BACKGROUND

```
setbg 111
```

JUST A LIGHTER SHADE

In case you don't like violet, there is a complete LYNX Colour Chart at the end of this book. Or you can use a graphical background, as we did in Project 3. Otherwise, you can get rid of the turtle, you won't need it in this project. Right-click on the **turtle** to open its dialog box and click on the garbage can.



## Create a text box for your encrypted message

Create a large text box on your page. Click on the **+** menu, choose **Text**. Right-click on it to open its dialog box and name it **MyText**.



Type one or two long sentences in the text box. This is called “clear text”, a text that anyone can read.

## Pseudo-code

Think of a way to scramble the message. In this first example, you will insert an “n” after each character (letter).

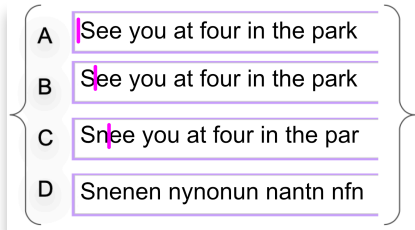
### WHAT IS PSEUDO-CODE?

Thinking in pseudo-code means that you make the program in your head (or on paper), using your own words, instead of real LYNX instructions.

## Project 4 - Secret codes (i\*l\*o\*v\*e\*)

Here is an example of pseudo-code for inserting a “n” after each character:

- A. Place the cursor at the beginning of the message.
- B. Move the cursor “one character” to the right.
- C. Insert an “n”.
- D. Repeat steps **B** and **C** until you reach the end of the message (you should figure out how many characters there are in the message)



### Make a procedure to encode the text

The pseudo-code sounds good. Now put that into real LYNX code!

In LYNX, you can give commands to the cursor (the insertion point) that's inside a text box, just like you can give commands to the turtle.

Here are the commands you will need for this secret code:

- **top** MOVES THE INSERTION POINT TO THE TOP OF THE TEXT BOX.
- **cf** (STANDS FOR CURSOR FORWARD)  
MOVES THE CURSOR ONE CHARACTER TO THE RIGHT.
- **insert 'n'** INSERTS WHATEVER LETTER, NUMBER OR SYMBOL YOU WANT, EXACTLY WHERE THE INSERTION POINT IS.

The other important trick is that the **name** of the text box (**MyText**) returns the entire contents of the text box, as a long, long word. Type this in the Command Centre:

**show mytext**

In our example, **mytext** reports **See you at four in the park** as one word.

You can use the primitive **count** to figure out how many characters there are in that long, long word, which is to say, how many characters there are in the text box:

**show count mytext**

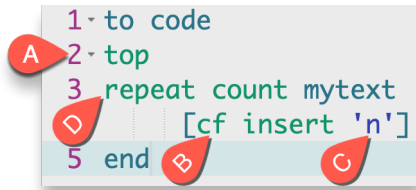
**27**

THIS IS WHAT LYNX RETURNS FOR OUR EXAMPLE

That is exactly how many times you have to **repeat** steps **B** and **C** in the pseudo-code above.

Now do you see how you can turn your pseudo code into real LYNX code? We will insert the letter *n*. You can choose any letter.

A = top, B = move cursor, C = insert 'n' and D = repeat [BC]



The instruction **repeat count mytext** is the same as **repeat 27** times (to match the number of characters in the **MyText** text box) the instructions inside the square brackets. It always gets the right number regardless of the message you type.

Try the procedure from the Command Centre:

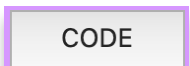
#### code

For your convenience, create a button to run the procedure. You have done this before. Here's a quick recap.

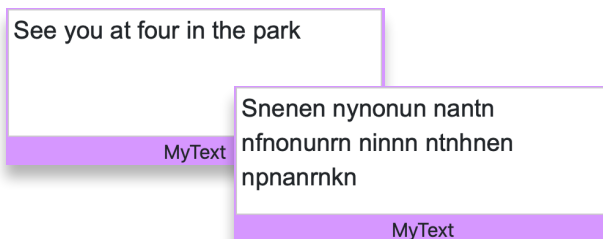
Click on the **+** menu then choose **Button**.

A button (**Nothing**) appears in the centre of the page. Right-click on it to open its dialog box.

In the dialog box that appears, type **CODE** as the label (write anything you like), and choose the procedure **code** in the **On Click** menu. Click **Apply**.



Type some new text in the text box and test the new button.



## Project 4 - Secret codes (i!\*o\*v\*e\*)

### Nnenendn ntnon ndnencnondnen nnnonwn?n

Do you understand the title above? Hard, eh! **Need to decode now?**

**First some pseudo-code:** Look at your encoded message. Think about what you DID to the message to encode it, and figure out a way to “undo” this.

- Bring the cursor to the beginning of the message.
- Move the cursor “1 character” to the right.
- Delete one character (the 'n' in our example).
- Repeat steps **B** and **C** up to the end of the message (**CAREFUL HERE**. The **encoded** message is **TWICE** as long as the **original** message, because you added lots of the letter “n”)

You will need one more command here:

**delete**

DELETES ONE CHARACTER TO THE RIGHT OF THE CURSOR MUCH LIKE THE DELETE KEY ON YOUR KEYBOARD

Create this procedure in the Procedures Pane:

```
7 ▾ to decode
8 ▾ top
9  repeat (count mytext) / 2
10  ..... [cf delete]
11  end
```

You see the first input of **repeat**?

The original message had **27** characters. You added 'n' 27 times to scramble the text, so now **count mytext** will say you have **54** characters in the text box. Want proof? Type **show count mytext** in the Command Centre. Divide that by **2** in order to delete just the **27** 'n'.

#### WHY THE “( )” AROUND “COUNT MESSAGE”?

They are needed, because without them, LYNX will try to divide the message (the long word) by two. That won't work! With the parentheses, LYNX will FIRST count the message (that will be a number), and THEN divide that number by 2.



Create a button in the usual way and give it a label. Choose **decode** in the **On Click** menu.

Type a new message and test *both* buttons.

Code

Decode

## Make it harder to decode!

You can train your eye to “detect” the character that was inserted (“n” in this case) and ignore it as you read. Instead of always inserting the *same* letter, try this:

Select the procedure **code**, **copy** it, click somewhere at the end of your procedures, and **paste** it. You now have two copies of the procedure **code**. **Warning:** you cannot have two procedures with the same name! Change the name of the **second** procedure to **better.code**, like this:

```
1 ▾ to code
2 ▾ top
3 repeat count mytext
4   ... [cf insert 'n']
5 end
```

```
13 ▾ to better.code
14 ▾ top
15 repeat count mytext
16   ... [cf insert 'n']
17 end
```

Now change the **better.code** procedure from this... to that.

```
13 ▾ to better.code
14 ▾ top
15 repeat count mytext
16   ... [insert 'n']
17 end
```

```
13 ▾ to better.code
14 ▾ top
15 repeat count mytext
16   ... [insert pick 'mwelun']
17 end
```

Instead of always inserting 'n' (procedure on the left), you can ask LYNX to **pick** a random letter and give it to **insert** (procedure on the right). Your poor eyes will be in pain now:

Swenen lyuoeeu uawtm nfnonulrl  
lienl utehuem uplalrlkw

MyText

In this example, we asked LYNX to randomly pick one of these six letters 'mwelun' and insert them into your text. These letters are harder to “ignore” than one letter like “n” or “z” or “x” so they make the encoded text even harder to read.

Create a new button for your **better.code** procedure.

Code

Decode

Better Code

# Project 4 - Secret codes (i!\*o\*v\*e\*)

How to use this project with your friends

1. Tell your friend “*I will send you a message. Delete every second character to decode it*”.
2. Use your LYNX project to encode a message.
3. Copy and transmit the encoded message to your friend - by email, text...
4. Or you can share your LYNX project with your friend. See *Appendix F* to learn how.



Pop Quiz: For your **better.code** procedure, do you think your **decode** procedure needs to change?

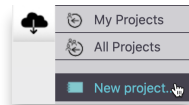
## Mission Impossible: Next level Secret Codes

When you encode a message, you don't have to hide it as you hand it to your friend, because nobody will understand the text. Here's a different approach. The message is **NOT** encoded, but it is simply *invisible* unless you have the correct password, and it will self-destruct after being read.

Ready for a different spy adventure? Save your current project - the small **red dot** indicates that there is something to save.



Then choose **New project** in the **Down from the cloud** menu:



And again, start by giving a new name to this project.

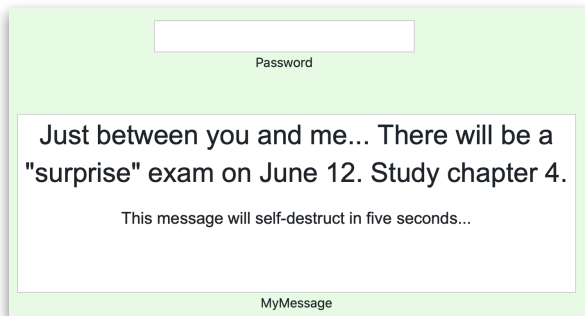


If you wish, make a nice background (use **setbg** to get a coloured background).

Create a **large** text box for your message, name it **MyMessage**.

Above it, create a **smaller** text box for entering the password, name it **Password**. Again, always use a single word, no space (not even at the end).

The **large** text box is where you will type the message you wish to “hide”. Here is an example of three lines you could type in the text box:



Let's try things “manually” before we create procedures and buttons.

Type something in the **MyMessage** text box.

Then type this in the Command Centre:

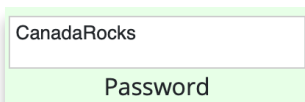
```
MyMessage, hidetext
```

IT IS IMPORTANT TO INCLUDE **MyMessage**, OTHERWISE, YOU COULD HIDE THE WRONG TEXT BOX!

**MyMessage** is now invisible. Don't worry!

## NOW ABOUT THE PASSWORD...

Type **CanadaRocks**, all one word, in the **Password** text box:



Then type this in the Command Centre:

```
show password
```

**CanadaRocks**

**PASSWORD** IS THE NAME OF THE TEXT BOX  
IT REPORTS THE CONTENTS OF THE TEXT BOX

And now, type this:

```
show password = 'canadarocks'
```

**true**

IS THIS EQUAL TO THE  
CONTENTS OF THE TEXT BOX?

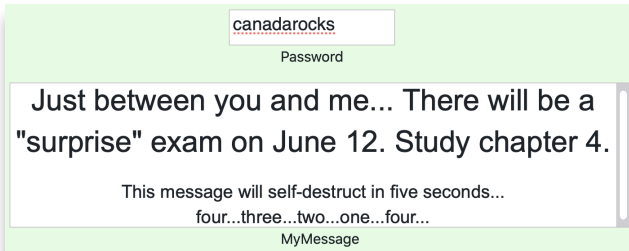
LYNX reports **true** because the contents of the text box (what is reported by **password**) is really equal to **'canadarocks'**. Casing (uppercase, lowercase) does not have to match.

## Project 4 - Secret codes (i!\*o\*v\*e\*)

And finally, try this, again in the Command Centre:

```
if password = 'canadarocks' [MyMessage, showtext]
```

The text box should reappear:

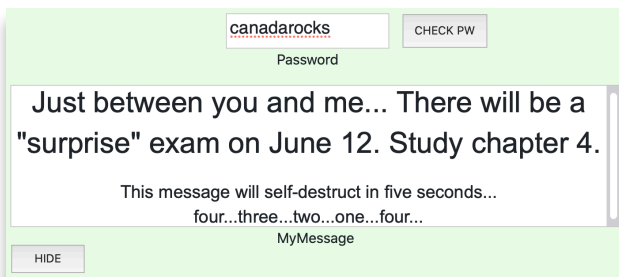


OK, you have all the pieces now. Let's code!

Imagine this scenario in your head:

- You tell your friend (in person or by text): “use **canadarocks** to reveal my secret” (this is sooo spy-talk)!
- You open this project and type a secret message in the large text box named **MyMessage**.
- You click on a **HIDE** button to hide the message.
- You share the project with the invisible message (see *Appendix F*) with your friend. Even if the project is intercepted, people who don't have the password won't be able to see the message!
- Your friend enters the password in the **Password** text box and clicks on **CHECK PW**. If the password is good, the message appears for 5 seconds, then *vanishes* just like in movies!

Here are all the parts you are about to create:



## OK, LET'S GET STARTED.

Create this procedure in the Procedures Pane (**cleartext** will clear the current text box, the one that is listening to your commands):

```
1 to hide
2 password, cleartext
3 mymessage, hidetext
4 end
```

Then create a button, with **Hide** as the label, and choose **hide** in the **On Click** menu.

HIDE

Resize and relocate the button if you wish.

Try the button. The **Password** box will be cleared of text, and the **MyMessage** text box will disappear. To get it back, either type this in the Command Centre:

**MyMessage, showtext**

or use the **Project Tree** to find the text box, select it and edit its dialog box. Check the box **Visible**.



Next, create the procedure that will be executed if your friend has the right password. It looks complicated, but it's not:

```
10 to display
11 mymessage, showtext
12 MyMessage, bottom
13 wait 10 setbg bg + 10 insert 'four...'
14 wait 10 setbg bg + 10 insert 'three...'
15 wait 10 setbg bg + 10 insert 'two...'
16 wait 10 setbg bg + 10 insert 'one...'
17 wait 10
18 mymessage, hidetext
19 password, cleartext
20 end
```

- **Line 11:** If you have the right password, I'll show you the text box with the message.
- **Line 12:** Going to the very end of the text, where it says "... will self destruct in five seconds..."
- **Lines 13 to 16:** set the background colour to "the current colour + 10", and insert '4...' then 3, 2, 1.
- **Line 18:** Hide the message.
- **Line 19:** Clear the password.

## Project 4 - Secret codes (i\*l\*o\*v\*e\*)

To test this, click on your **HIDE** button, and type this in the Command Centre:

**display**

The message should reappear, and you should see some flashing of the background.

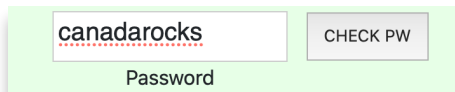
Last but not least, code a procedure to validate the password.

Remember, the people with whom you share your project won't see the Procedures Pane (your code contains the actual password) and they won't have a Command Centre. So you have to provide a procedure and a button to validate the password:

Create this procedure...

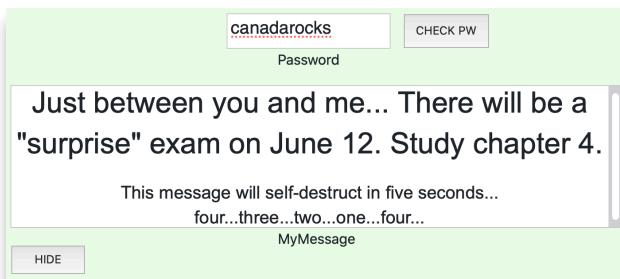
```
7 to check.password
8 ; change the password here
9 if password = 'canadarocks' [display]
10 end
```

and create a button to run it:



### TEST, AND SHARE

Do you have all these elements?



Test everything: Type a message, click **HIDE** to hide it, try a bad password (and click **CHECK PW**), then try the good password.

When sharing (see *Appendix F* for instructions), make sure you keep the project **Private**, so people won't see your code (and password).

Remember, you can edit your own code and decide on a new password at any time!

## All the procedures of this project

To help you, here are all the procedures you need. To explain things we have included comments which begin with a `;` and are in grey. Your code and comments can be different, of course!

### ENCODING A MESSAGE

```
to code
top
repeat count mytext
    [cf insert 'n']
end

to decode
top
repeat (count mytext) / 2
    [cf delete]
end

to better.code
; works for code and better.code
top
repeat count mytext
    [cf insert pick 'mwelun']
end
```

### HIDING A MESSAGE

```
to hide
password, cleartext
mymessage, hidetext
end

to display
mymessage, showtext
mymessage, bottom
wait 10 setbg bg + 10 insert 'four...'
wait 10 setbg bg + 10 insert 'three...'
wait 10 setbg bg + 10 insert 'two...'
wait 10 setbg bg + 10 insert 'one...'
wait 10
mymessage, hidetext
password, cleartext
end

to check.password
; change the password here
if password = 'canadarocks' [display]
end
```

# Project 4 - Secret codes (i\*!l\*o\*v\*e\*)

## More advanced ideas for super spies

### ENCODING A MESSAGE

Here are some ideas for different encryption recipes. Can you code them? Look exactly at what is being inserted, and how the cursor is moved:

Recipe	Decode method
repeat count mytext [insert 'un' cf]	Move <b>one</b> character to the right and delete <b>two</b> characters
repeat count mytext [insert pick 'wmunei' cf cf]	Delete every <b>third</b> character
repeat count mytext [insert 'u' cf insert 'n' cf cf]	???

### MORE FUN WITH THE COUNTDOWN

In the last project, during the countdown before the message disappears, the background changes colour every second. Think of other ways to make the countdown fun:

- you add a sound, for example, a paper crumpling or burning.
- a different text box shows 5, 4, 3, 2, 1, 0.
- the computer reads that text box at every second.

## Curriculum Links for Ontario

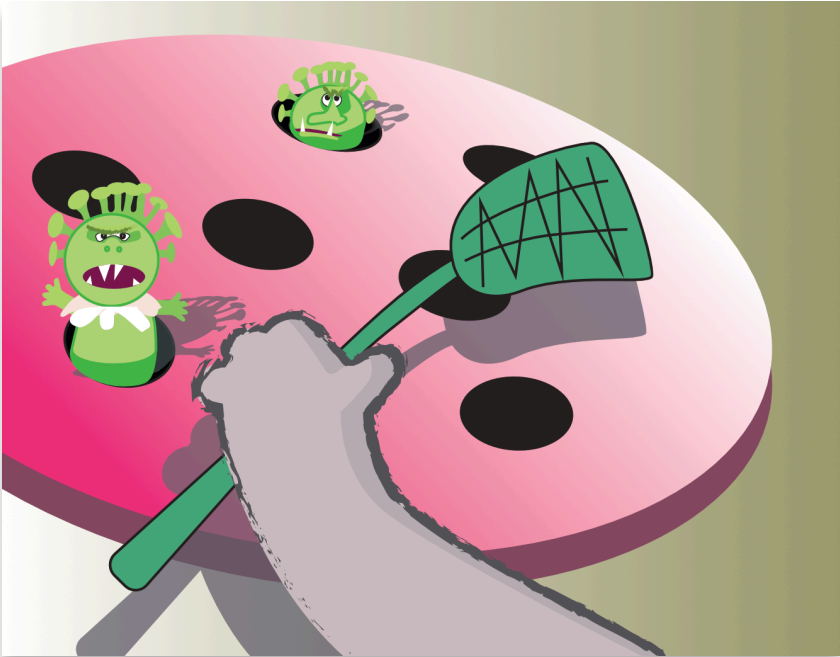
**C3.1** - Solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves conditional statements and other control structures.

**C3.2** - Read and alter existing code, including code that involves conditional statements and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code.



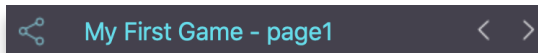


## Project 5 - Kill the virus



Many years ago there was a game called Whack-a-mole. Moles appear **randomly** and very **briefly** on a board or in an arcade game machine and you have to hit them before they go back underground. Let's modify it for the reality of COVID-19. Let's program a video game and call it *Kill the Virus*. Our game will involve **animation**, **score keeping**, and **controls** (to make the game harder).

Start a new project, and give it a new name now. Don't forget to save often!



## A plain background


First, select a nice background for the game. It can be as simple as a coloured background (use the `setbg` command, check the colour chart in *Appendix C*). Or, find a clipart image of a protective mask and import it into your Clipart Pane.

If you chose to use a `setbg` background, you probably still have your turtle on the screen, waiting for instructions. Perfect.

If you chose to use a stamped image background, don't forget to **stamp** the large image, and set the turtle back to its original turtle shape using **setshape 0**.

## Find a virus clipart

Time to get a clipart for the virus popping up. Look in the **Sample Clipart / Nature, Other** for something you could use. Otherwise, look in *Appendix D* for clipart resources on the Web, and special instructions regarding clipart with transparent surrounding. In a few words:

- Find a PNG image with transparent surrounding. Often seen on a grey or a checkered background.
- Download the image to your computer.
- Open the Clipart Pane, click on a free spot, then on the  in that free spot.
- This opens the Import image dialog box. Navigate to the image that you have just downloaded. You now have your clipart!



## Change the turtle to the virus

Use the virus clipart to change the turtle's shape, and set the size of the turtle-virus, not too big, not too small. You can always adjust this detail later if the game is too easy or too difficult.

```
setshape 2  
setsize 10
```

USE YOUR OWN CLIPART NUMBER  
PLAY WITH DIFFERENT NUMBERS

Then, create a procedure that the virus will execute when you "hit" it.

```
to hit  
; grow big for a split second, then back to normal  
virus,  
setsize 15  
wait 3  
setsize 10  
end
```

YOU ARE ABOUT TO RENAME THE TURTLE FROM T1 TO VIRUS

FYI: WAIT 10 IS 1 SECOND

Later on, you will add to this procedure, to keep the score of the good hits. But for the moment, right-click on the virus to open its dialog box:

## Project 5 - Kill the virus

Plenty of things to do and learn here! Let's start with just three things:

a) First, choose the **hit** procedure in the **On click** menu:

The dialog box is titled 'Hit' and contains the following fields and options:

- Name:** A text field containing 'VIRUS'.
- Xcor:** A text field containing '52'.
- Ycor:** A text field containing '52'.
- On click:** A dropdown menu with 'hit' selected.
- On touch:** A dropdown menu with '-' selected.
- On message:** A dropdown menu with '-' selected.
- On colour:** A dropdown menu with '-' selected.
- Visible:** A checked checkbox.
- Frozen:** An unchecked checkbox.
- Buttons:** 'Apply' and 'Cancel' buttons at the bottom.

b) At the top of the dialog box, you see the name of the turtle (**t1**). When you add turtles to the page, the turtles are named **t1**, **t2**, **t3** and so on. You can change the name, but always use a single word, no space. Change the name to **VIRUS** in order to match the name in the **hit** procedure you just made.

c) Then you see its current **x** and **y** coordinates positions. More about these in a moment.

Click on **Apply** to save your changes, and give it a try: Click on the virus, it should grow and shrink, as instructed in the **hit** procedure.

### Play procedure

Time to pseudo-code. What does it mean to “play” this game. Think about it before continuing to read.

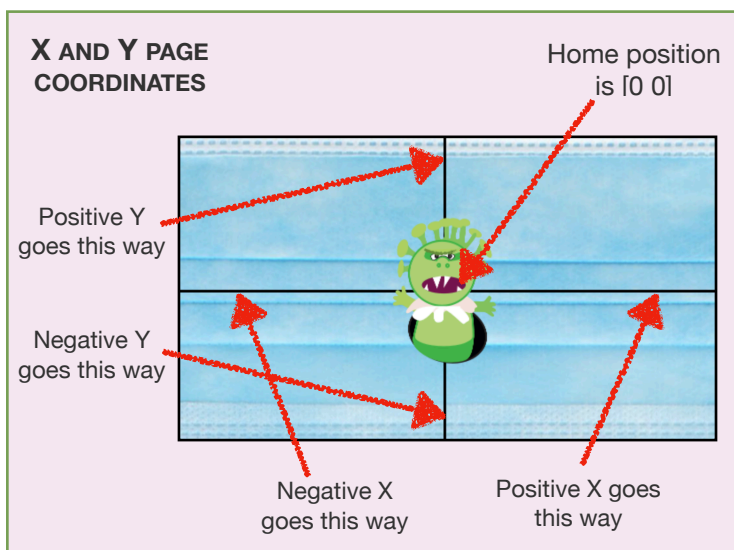
- Hide the virus.
- While hidden, move the virus to a random place.
- Make the virus appear for a very brief moment.
- If the user manages to click on the virus while it is visible, run the **On click hit** procedure.

The key action here is “move the virus to a random place”. Let’s see how you can make this happen. Type this in the Command Centre:

```
home
show pos
0 0
setx -200
setx 200
setx 0
sety 200
sety -200
sety 0
```

```
THE VIRUS GOES TO THE CENTRE OF THE PAGE
SHOW ME YOUR POSITION
THE CENTRE COORDINATES [X Y] ARE [0 0]
MOVE TO THE X COORDINATE -200
MOVE TO THE X COORDINATE 200
MOVE TO THE X COORDINATE 0
DO THE SAME FOR THE Y COORDINATE
```

What have we learned here...



If you are wondering how much right-left, top-down you can go, type this in the Command Centre:

```
show projectsize
800 450
```

This is the size of a **standard** project (see footnote <sup>1</sup> below). You can select a different size when you start a new project. In this example, the **maximum X** and **Y** positions are -400 and +400 horizontally (for **X**), -225 and +225 vertically (for **Y**).

<sup>1</sup> To create a project of a different size, choose New project in the **Down from the cloud** menu (arrow down from the cloud). Choose a size in the dialog box (standard, large, small, custom).

## Project 5 - Kill the virus

These are the limits for the **page**, for this **project size**. We will make the **play area** just a bit smaller: -300 to 300 horizontally (**X**), and -200 to 200 vertically (**Y**).

### MOVE THE VIRUS TO A RANDOM POSITION

This game is based on the fact that the virus will move to a random position on the screen before appearing briefly. Here's what to do:

Try this in the Command Centre:

```
setx 300          THE VIRUS GOES EXACTLY TO THE X COORDINATE 300
setx random 300   A RANDOM X COORDINATE, LESS THAN 300
setx random 300   ANOTHER RANDOM X COORDINATE, LESS THAN 300
```

**Random 300** will always give you a positive number (between 0 and 299), and using that for **setx**, the virus will always end up on the right-hand side of the page. Too easy to hit, right! What you really need is a random number between say... **-300** and **300**. To get this result, take **-300**, and add a **random number between 0 and 601**. The result will always be between **-300** and **300**. Just imagine:  $(-300 + 0)$   $(-300 + 1)$   $(-300 + 2)$  up to  $(-300 + 601)$ . Type this in the Command Centre and try it a few times:

```
setx -300 + random 601    RANDOM 601 REPORTS A NUMBER
                           BETWEEN 0 AND 600 BUT WE
                           STARTED ON THE LEFT EDGE AT -300
```

With that in mind, create this **move.show** procedure in the Procedures Pane:

```
to move.show
; hide, move, show for a short while, hide again
; play area is 600 x 400
virus,          OR WHATEVER NAME YOU USED FOR T1
ht
setx -300 + random 601
sety -200 + random 401    SAME LOGIC MUST APPLY TO THE Y AXIS
st
wait 8
ht
end
```

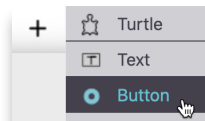
And a **play** procedure, for the game to keep going:

```
to play
  forever [move.show]
end
```

Create a button and set the **On Click** action to the procedure **play** as follows:

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

Right-click to open its dialog box, type a label, and set its instruction to **play**.

A screenshot of the 'Button' dialog box. It has three fields: 'Name' with the value 'button1', 'Label' with the value 'PLAY', and 'On click' with a dropdown menu showing 'play'.

Make sure you put the button in a far corner, away from the main play area.



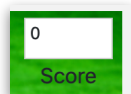
Test the button: the virus should appear in random places. Try to hit it when it shows itself. When hit, the virus should grow and shrink.

## Let's record the number of hits

Time to keep score! You can record the score in a text box.

Click on the **+** menu then choose **Text**.

A text box appears on the page. Type **0** in it. Make it quite small and move it to a corner of the page. Then, right-click on it to open its dialog box, and rename it **score**.

A screenshot of the 'Text' dialog box. It has a 'Name' field with the value 'Score'. There are two checked checkboxes: 'Show name' and 'Visible'. There are two unchecked checkboxes: 'Transparent' and 'Frozen'. At the bottom, there are 'Apply' and 'Cancel' buttons.

Try this in the Command Centre:

```
show score    SHOW ME THE CONTENTS OF THE TEXT BOX NAMED SCORE
0             CURRENT CONTENT IS 0
setscore score + 1  SET THE CONTENT TO "CURRENT VALUE" + 1
show score    SHOW AGAIN PLEASE
1             CURRENT CONTENT IS NOW 1
```

## Project 5 - Kill the virus

See, **score** reports the contents of the text box with that name, and **SETscore** sets the contents of the text box with that name. That's how you increase the score.



When exactly do you need to increase the score? In which procedure?

Right! When you manage to hit the virus. Add this to the hit procedure:

```
to hit
; grow big for a split second, then back to normal
setscore score + 1
virus,
setsize 15
wait 3
setsize 10
end
```

Score keeping requires two new actions for your game to work. At the beginning of the game, you must reset the score to 0. And each time the score increases, you have to check if the player is a winner (when the top score or maximum hits permitted are reached).

### A Reset procedure and button

A reset procedure is a simple thing. Create this procedure:

```
to reset
setscore 0
virus, ht
end
```

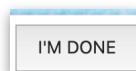
HT STANDS FOR HIDE TURTLE

Create a button to reset the game, place it near the other button.

Try the **RESET** button, then the **PLAY** button, and after a few hits, click on **PLAY** button to stop the game and try **RESET** again.



Now YOU have access to the **Stopall** button in the Command Centre, but if you **share** the project with friends, there won't be a Command Centre. Create this **stop.now** procedure and a button that goes with it. Next add a button and choose **stop.now** in the **On Click** field.



```
37 to stop.now
38 stopall
39 end
```



## Announce the champion!

Time to reward the champion. You remember the procedure where you increase the score? Add a conditional instruction: if the score reaches 10, announce the winner and stop the game.

Edit the procedure `hit`:

```
to hit
; grow big for a split second, then back to normal
setscore score + 1
if score = 10 [winner]
virus,
setsize 15
wait 3
setsize 10
end
```

And now make a `winner` procedure. It uses the primitive `announce`, which causes a message to appear on the screen.

```
to winner
announce [Congrats! You killed the virus 10 times]
stopall
end
```

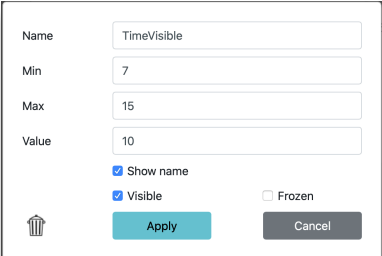
Don't click on **RESET**, type **8** in the text box (just cheating to complete the game faster), and click on **PLAY**. After two hits, you get to **10** and you should get a big message and the game should stop.

## Add different levels of difficulty


Many games start “easy” and increase the level of difficulty as you go. What would make this game easier or more difficult?

The brief moment when the virus is visible could be longer in the beginning, and shorter later on.

Click on the **+** menu, choose **Slider**. Right-click on the slider and name it **TimeVisible** (one word, no space). Set the minimum at 7, and the maximum at 15 and the current value at 10. You can change these values later if you are not happy with them. Place the slider near the top or bottom edge of the page.



The screenshot shows a configuration window for a slider control. It has the following fields and options:

- Name:** TimeVisible
- Min:** 7
- Max:** 15
- Value:** 10
- ☒ Show name
- ☒ Visible
- ☐ Frozen
- 
- Apply** button
- Cancel** button

## Project 5 - Kill the virus

You can use the slider just like you did for the **Score** text box. The slider name reports its value. The word **set**, followed by the name of the slider, can be used to set its value. Try these instructions in the Command Centre:

```
show timevisible
```

SHOW ME THE CURRENT VALUE OF THE SLIDER

```
10
```

THIS ANSWER DEPENDS ON YOUR SLIDER

```
settimevisible 15
```

SET THE SLIDER'S VALUE TO 15

```
settimevisible timevisible - 1
```

LOWER THE VALUE BY 1



When should you decrease the value of the slider to reduce the time the virus is visible and make the game more difficult?

Yes, when the player manages to hit the virus. Edit the **hit** procedure again:

```
to hit
  setscore score + 1
  if score = 10 [winner]
  if timevisible > 7 [settimevisible timevisible - 1]
  virus,
  setsize 15
  wait 3
  setsize 10
end
```

### WHY THE CONDITION "IF TIMEVISIBLE > 7" ?

The slider cannot go below 7, the lower limit that you have set. If you try to go below, LYNX will display an error message. So when you hit the virus, the slider will go lower ONLY if it is 8 or above.

Finally, you should also reset the slider's value when you reset the game. Edit the **reset** procedure:

```
to reset
  setscore 0
  settimevisible 15
  virus, ht
end
```



Where is the right place to use this value? In your procedures, where do you manage how long the virus is visible?

Yes, it is in the `move.show` procedure, where it says “show, wait a bit, hide”. Instead of using a fixed number for `wait` use the value of the slider:

```
to move.show
virus,
ht
setx -300 + random 601
sety -200 + random 401
st
wait timevisible
ht
end
```

HT STANDS FOR HIDE TURTLE

ST STANDS FOR SHOW TURTLE

## Testing and sharing

Time to test everything. Test it in the **LYNX editor** first, so you can make corrections, then test it in **Play mode** (on the page **My Projects**) to see the game as *your friends will see it*.

- Click on **RESET**. Is the text box reset to 0? Is the slider reset to 15?
- Click on **PLAY**. Is the virus moving and showing? Is it easy at the beginning? Is the slider decreasing when you hit the virus?
- Can you stop the game by using the button **I'M DONE**?
- Is the game playable (not too difficult)? What values can you change to make it easier or harder?

Did everything work as you expected? Ready to share this game with your friends? See *Appendix F* about sharing.



Is the game too easy? Too difficult? Would you know how to fix that? What about setting the `timevisible` to a larger number at the start (`reset`) so the game is easier for very young children?

# Project 5 - Kill the virus

## All the procedures of this project

Just in case you need to check something, here are all the procedures in the project, along with some comments in grey. Your comments can be different, of course!

### to hit

```
; grow big for a split second, then back to normal
setscore score + 1
if score = 10 [winner]
if timevisible > 7 [settimevisible timevisible - 1]
virus,
setsize 15
wait 3
setsize 10
end
```

### to move.show

```
; hide, move to a random position
; show for a short while hide again
; play area is 600 x 400
virus,
ht
setx -300 + random 601
sety -200 + random 401
st
wait timevisible
; timevisible is the value of the slider
ht
end
```

### to play

```
forever [move.show]
end
```

### to reset

```
; text box and slider
setscore 0
setTimeVisible 15
virus, ht
end
```

### to stop.now

```
; need a stopall button to halt the game before its
normal ending
stopall
end
```

```
to winner
```

```
; 10 hits and it's game over... for the virus!  
announce [Congrats! You killed the virus 10 times!]  
stopall  
end
```

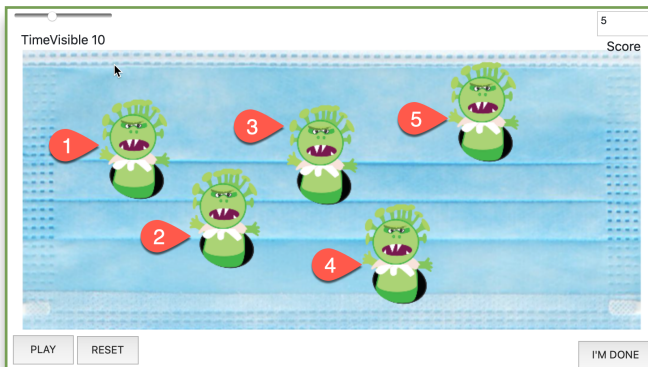
## More advanced ideas

There is a totally different way to code this game. You can have five *fixed* viruses, in five different locations, *never* moving around the page. Your **play** procedure would not do **forever [move.show]**, but **forever [pick.show]**. In this example, we will keep the original turtle names (t1, t2...) **Pick.show** would be something like:

```
to pick.show  
ask pick [t1 t2 t3 t4 t5] [st wait timevisible ht]  
end
```

The trick here is **ask**. Ask needs two inputs: which turtle is being *asked*, and what it is being asked to do. You need to use square brackets **[ ]** around each input.

- The first input will be a random turtle-virus  
**pick [t1 t2 t3 t4 t5]**.
- The second input (**[st wait timevisible ht]**) is the “short appearance” of the turtle, same as the **move.show** procedure.



In this image, there are five turtles, named **t1** to **t5**. They will appear briefly one at a time, randomly picked by LYNX.

## Project 5 - Kill the virus

You can prepare a “master” turtle-virus and then clone it 4 times. All the properties of the master turtle-virus will transfer to the new turtle-viruses. Run this instruction 4 times:

```
clone 't1'
```

```
THE NAME OF THE TURTLE YOU WANT TO CLONE
```

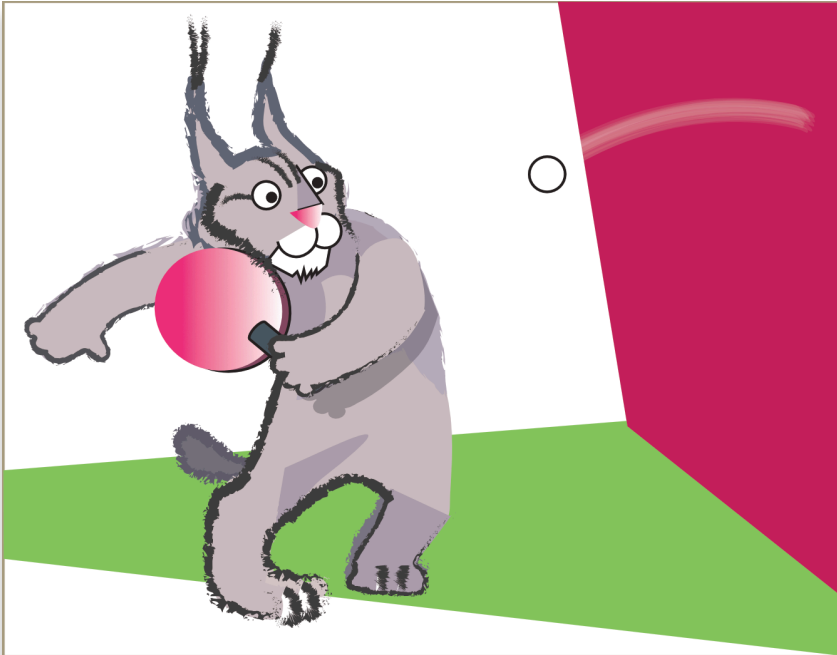
### Curriculum Links for Ontario

**C3.1** - Solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves events influenced by a defined count and/or sub-program and other control structures

**C3.2** - Read and alter existing code, including code that involves events influenced by a defined count and/or sub-program and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code



## Project 6 - Pong!



Another classic game! In this version of Pong, you will play against the computer. You will control the paddle, and the ball will bounce on a wall on the other side of the page. This advanced project introduces **colour detection**, **collision detection**, and **mouse interaction**. Let's start! You are, after all, making a game that your friends can play!

Start a new project, and give it a new name now. Don't forget to save once in a while!



### Create a ball

First, you need a ball. You can draw one using any paint program you may have, and save it as a PNG file *with transparency* around the ball. Or, you can download a ball clipart from a web site (see *Appendix D - Finding Things*). Look for a small file size, under 50 K.

If you don't use a PNG file with transparency, the ball will be round, but it will look like a white block when you place it on a nice background. Read more about this in *Appendix D*.



When you have a ball image, it's time to import it into your project:

- Open the Clipart Pane.
- Click anywhere in an empty spot.
- Click on the “+” to open the Import dialog box

A screenshot of the 'Import image...' dialog box. It has a title bar with a close button (x). Inside, there is a label 'URL' followed by a text input field containing the word 'URL'. Below the input field are two buttons: 'Select...' and 'Create'.

- Click on **Select** to locate your your Ball file on your computer, then click on **Create**.
- There, you have it! Remember the number for the clipart.



## A BALL-TURTLE

If you don't have a turtle on the page, create one now. Choose **Turtle** in the **+** menu. Set its heading to 75 degrees. Type this in the Command Centre:

**setheading 75**

OR USE SETH 75

Give it the shape of a ball. You can either run this instruction in the Command Centre:

**setshape 1**

USE YOUR CLIPART NUMBER

or click on the Ball clipart to turn the mouse pointer into a hand, then click on the turtle. Click and Click, not Click and Drag!

Your clipart image may be too large. If the ball looks huge on the page, use an instruction such as this one to change its size:

**setsize 10**

LESS THAN 40 TO MAKE IT SMALLER,  
MORE THAN 40 TO MAKE IT LARGER.

## GET THAT BALL MOVING!

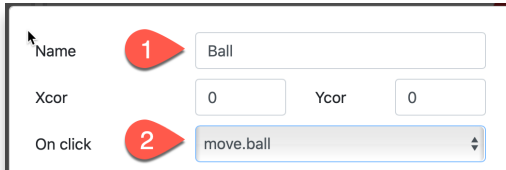
Create this procedure to make the ball glide in the play area. You will code the bouncing later.

```
to move.ball
ball,
forever [forward 10 wait 1]
end
```

# Project 6 - Pong!

Now right-click on the ball to open its dialog box.

- 1) name this turtle **Ball** (remember, no space, not even before or after the word), and
- 2) choose **move.ball** in the **On click** menu.



The screenshot shows a dialog box for a turtle named 'Ball'. It has three fields: 'Name' with the value 'Ball', 'Xcor' with the value '0', and 'Ycor' with the value '0'. The 'On click' field is a dropdown menu showing 'move.ball'. Red callout boxes with numbers 1 and 2 point to the 'Name' and 'On click' fields respectively.

Close the dialog box and click on the ball to test it. You can always come back to this procedure to change the speed of the ball. You know what number to change for speed, right?

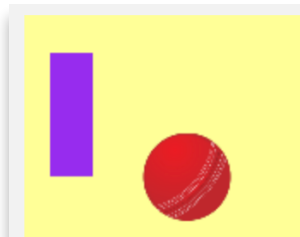
## Create a paddle

### A PADDLE-TURTLE

The routine is similar to the ball routine. Use any paint program, or take a screenshot. This time, since the paddle is a full rectangle, the transparency around the paddle is not an issue.

- Create the clipart file.
- Load the clipart into an empty spot in the Clipart Pane. Remember you can also Copy the clipart file and Paste it into an empty spot.
- Create a new turtle.
- Give the paddle clipart to the turtle using **setshape** or Click and Click.
- Resize the paddle-turtle if needed.

At this point, you should have a ball and a paddle on the page.



## CONTROLLING THE PADDLE

This is the key element of the game: in order to play, the paddle must “follow” your mouse pointer. Right now, let’s do this in a simple way - the Going Further section, at the end of the chapter, contains suggestions to improve the interaction.

Type this in the Command Centre:

```
show mousepos
```

```
-312.75 -497.671875
```

MEANS MOUSE POSITION, PRESS ENTER

You will get a set of two numbers, the X and Y coordinates, based on the position of your mouse pointer on the screen. Your numbers will be different.

Now type this in the Command Centre:

```
forever [setpos mousepos]
```

Move the mouse pointer across the work area. The paddle should be following you, as long as you stay in the page.

You see, **mousepos** reports a set of two numbers, and **setpos** uses these numbers to set the turtle’s position.

### IS THE BALL FOLLOWING THE MOUSE?

If the ball, not the paddle, is following the mouse, that’s because the ball is the current turtle. Stop everything, click on the paddle to “make it current”, i.e. the turtle that will listen to commands right now. Run the instruction again.

The current turtle is:

- (a) the last turtle that you have created, or
- (b) the last turtle that you have clicked on, or
- (c) the last turtle that you called using the “comma” method (`paddle,`).

Click on the **Stop All** button to stop the **forever** instruction.

Create this procedure to make the paddle always follow the mouse position while playing:

```
to move.paddle  
paddle,  
forever [setpos mousepos]  
end
```

# Project 6 - Pong!

Now right-click on the paddle to open its dialog box. 1) name this turtle **Paddle**, and 2) choose **move.paddle** in the **On Click** menu.



The screenshot shows a dialog box for a turtle named 'paddle'. It has four fields: 'Name' with the value 'paddle', 'Xcor' with the value '365', 'Ycor' with the value '162', and 'On click' with a dropdown menu showing 'move.paddle'. Red callout boxes with numbers 1 and 2 point to the 'Name' and 'On click' fields respectively.

## Bounce off the paddle

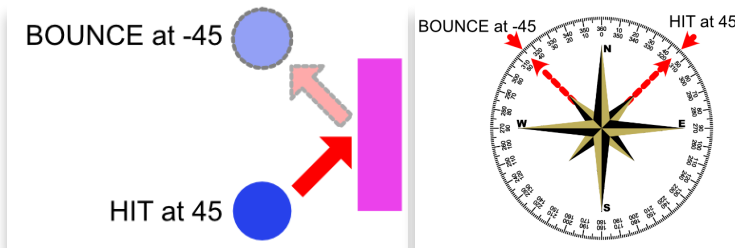
The ball and the paddle are both turtles. LYNX can detect collisions between turtles. When they collide, the ball should bounce.

There are many ways to code the bounce. It is all about the heading **before** the collision, and the heading **after** the collision. Here are five different possibilities that could happen to the ball after colliding:

```
rt 180
setheading minus heading
setheading heading + 180
setheading 360 - heading
setheading 180 + random 180
```

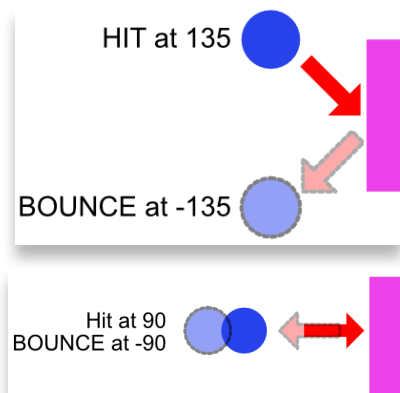
They are all worth trying and understanding, but we will focus on this one for the moment: **setheading minus heading**.

Consider this example: the ball is coming from the left and hits the paddle with a heading (on a compass) of 45 degrees:



On a compass, **0** degrees is **North**. **45** degrees is **North-East**. If you use negatives, you simply go counterclockwise: **minus 45** degrees is **North-West** - it is just the **mirror** of 45 degrees.

And that works for other angles too!



When should you execute the bounce procedure? When the ball hits the paddle! Right-click and open the ball's dialog box and choose **New** in the **On touch** menu:

Name	<input type="text" value="Ball"/>		
Xcor	<input type="text" value="37"/>	Ycor	<input type="text" value="111"/>
On click	<input type="text" value="move.ball"/>		
On touch	<input type="text" value="New..."/>		

This creates a procedure like this one in the Procedures Pane:

```
11 ▾ to Ball_touch :touchedturtle
12 ; Use instructions like these to set this turtle's
   ; reaction when it touches another turtle.
13 ; The variable :TOUCHEDTURTLE contains the name of the
   ; other turtle.
14 ; SAY "OUCH!
15 end
```

The grey text is just comments to help you understand Collisions. Read them and delete them. Then, fill in the procedure as follows:

```
11 ▾ to Ball_touch :touchedturtle
12   ball,
13   setheading minus heading
14   end
```

You don't have to worry about who is the touched turtle, it is always going to be the paddle *as it is the only other turtle in this game*.

# Project 6 - Pong!

The Ball's dialog box now looks like this, with both an **On click** instruction and an **On touch** instruction:

Name	Ball		
Xcor	37	Ycor	111
On click	1 move.ball		
On touch	2 ball_touch		

## FIRST BIG TEST

Set everything into action. Type this in the Command Centre:

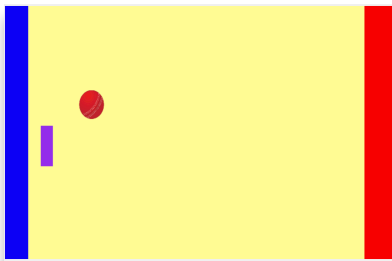
**everyone [clickon]**

**Everyone** means “every turtle on the page” and **clickon** means “act as if you got a mouse click”.

The ball should start gliding. Move the mouse around the page, the paddle should track the mouse pointer. Place the paddle in the path of the ball. Did you get a bouncing action? If yes, it is working!

## Build some walls

Here’s a plan before we start building walls.



- The play area has a paddle and a ball.
- The ball will bounce on the right (red) wall, and each time, you get a point.
- The ball heads left, and if it hits the paddle, it stays in play. If you miss, the ball will hit the left (blue) wall, and you lose a point.

- Your score starts at 10. If you hit 20 points, you win the game. But if you get down to 0, it's game over!

- Why not paint the background a nice colour? Use the LYNX Colour chart in *Appendix C* and remember the number. Then type this in the Command Centre:

- `setbg 42`

OR WHATEVER NUMBER YOU CHOSE

Here's a quick way to draw the walls.

- Create a new turtle.
- Drag it near the right edge of the page.
- Type this in the Command Centre:  
`setcolour 'red'`  
`pendown`  
`forward 9999`
- This will draw a thin red line as shown on the right.
- Now drag the turtle to the right of the red line, and execute this instruction:



`fill`

FILLS THE AREA WITH THE TURTLE'S CURRENT COLOUR

- Type `setcolour 'black'` to set the turtle back to **black**, so you can see it again (can't see red on red). You should have a red wall now.

Do the same for the left blue wall.

- Drag the turtle near the **left** edge of the page, and execute these instructions in the Command Centre:  
`setcolour 'blue'`  
`pendown`  
`forward 9999`
- Drag the turtle to the left of the blue line, and execute this instruction:  
`fill`

Do you have something like the image on the previous page?

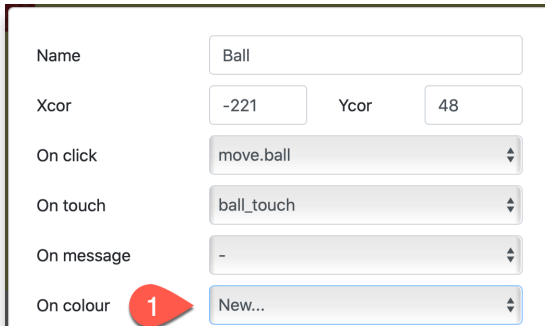
Type `setcolour 'black'` to set the turtle back to **black**, so you can see it again. Right-click on this turtle and use the garbage can to delete it.

# Project 6 - Pong!

## Count the good hits on the red wall

### COLOUR DETECTION

First, let's create the procedure that deals with colour detection. Open the ball's dialog box and choose **New** in the **On colour** menu.



This creates a procedure such as this one:

```
20 to Ball_oncolour :prevColour :newColour
21 ; Use an instruction like this to do colour detection
   every time
22 ; Moving from any colour to red, even red to red, will
   trigger the action.
23 ; Pick your own colour name and instructions instead of
   [BACK 10 RIGHT 180]
24 ; IF :NEWCOLOUR = "RED [BACK 10 RIGHT 180]
25 end
```

Again, the grey text is just comments. Read and delete them, then fill in this code:

```
20 to Ball_oncolour :prevColour :newColour
21 if :newColour = 'red' [hit]
22 if :newColour = 'blue' [miss]
23 end
```

**Hit** and **miss** are two procedures that **do not exist yet**. Don't worry, we're getting there. Before we do, let's examine **Ball\_oncolour**: When the turtle moves, it always checks what colour it is currently on. That is the **:newColour**. If it touches the red wall, the value of the variable **:newColour** will be **'red'**.

So if the **:newColour** is **'red'**, you run the **hit** subprocedure. If the **:newColour** is **'blue'**, you run the **miss** subprocedure.

The turtle-ball will not react to any other colour!



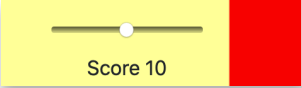
## A SLIDER TO COUNT THE HITS

How about this: you can use a slider to count the score. The slider goes from 0 to 20, and the game starts at 10. You win a point on the red wall, and lose a point on the blue wall.

In the **+** menu, choose **Slider**.

Move the slider to a corner of the page (drag it by its name), and right-click on it to open its dialog box. Name it **Score**, and set the **MINimum and MAXimum** values and the **current** value as follows:

Name	Score
Min	0
Max	20
Value	10



## WE HAVE A WINNER

Remember the instruction `if :newColour = 'red' [hit]`

Now we can work on the `hit` procedure. Create this procedure in the Procedure pane:

```
to hit
  setheading minus heading
  setscore score + 1
end
```

When the ball hits the red wall, it should bounce and increase the score. **Score**, the name of the slider, reports its current value.

**Setscore** sets its new value, to the old value, plus one.

# Project 6 - Pong!

## Count the misses on the blue wall

The miss procedure is very similar. Can you guess?

```
to miss
  setheading minus heading
  setscore score - 1
end
```

There is a problem ahead. You can try `everyone [clickon]` in the Command Centre. The ball will bounce off the paddle, and off both walls. However, if you reach 0 or 20, you will get an error message:

```
scorescore doesn't like 21 (or -1) as input
```

Naturally, because its limits are 0 and 20.

We need to change the `hit` and `miss` procedures. Add a line that checks the limit. The new line is highlighted:

```
to hit
  setheading minus heading
  setscore score + 1
  if score = 20 [announce 'You win!' play.again]
end
```

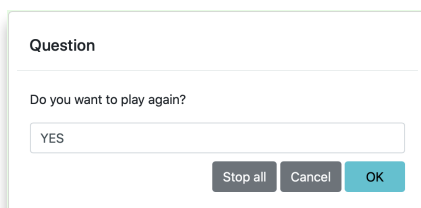
```
to miss
  setheading minus heading
  setscore score - 1
  if score = 0 [announce 'Game over!' play.again]
end
```

Let's check the case of the `hit` procedure. You hit the red wall. The ball bounces. The slider increases by one. Now check if the `score = 20`. If it is, announce the winner (that's what `announce` does), and run the procedure `play.again` (it doesn't exist yet).

The procedure `play.again` uses the primitives `question` and `answer`. Let's look at them. Type this in the Command Centre:

```
question [Do you want to play again?]
```

LYNX displays a dialog box with that question. Type **YES** and click **OK**.



Question

Do you want to play again?

YES

Stop all Cancel OK

Now type this in the Command Centre:

```
show answer      ANSWER REPORTS THE LAST ANSWER IN A DIALOG BOX
show answer = 'yes'
true             LYNX REPORTS TRUE OR FALSE
```

Create this procedure in the Procedures Pane:

```
to play.again
question [Do you want to play again?]
ifelse answer = 'yes'
  [start.game]
  [stopall]
end
```

You know about `question` and `answer`, but now look at `ifelse`. `Ifelse` checks if the answer was `'yes'`. If true, it runs the **first** list of instructions: `[start.game]`. If false, it runs the **second** list of instructions: `[stopall]`.

You don't have to format the code on three lines like that, but it makes all the instructions easier to read.

## A Start button

You need a button to start the game. Remember how Player mode works. When you share this game, there will be no Command Centre and no Procedures Pane.

Create this procedure:

```
to start.game
ball,
setpos [0 0]
setheading 75
setscore 10
everyone [clickon]
end
```

CHOOSE WHATEVER # YOU LIKE  
CHOOSE WHATEVER # YOU LIKE

At this point, you should be able to guess what each line does. Finally, create a button and set its **On click** instruction to `start.game`

# Project 6 - Pong!

## All the procedures of this project

```
to move.ball
ball,
forever [forward 10 wait 1]
end
```

```
to move.paddle
paddle,
forever [setpos mousepos]
end
```

```
to Ball_touch :touchedturtle
; bounce when ball touches paddle
ball,
setheading minus heading
end
```

```
to Ball_oncolour :prevColour :newColour
; red wall on right, blue wall on left
if :newColour = 'red' [hit]
if :newColour = 'blue' [miss]
end
```

```
to hit
; bounce, then increase slider, max is 20
setheading minus heading
setscore score + 1
if score = 20 [announce 'You win!' play.again]
end
```

```
to miss
; bounce, then decrease slider, min is 0
setheading minus heading
setscore score - 1
if score = 0 [announce 'Game over!' play.again]
end
```

```

to play.again
question [Do you want to play again?]
ifelse answer = 'yes'
  [start.game]
  [stopall]
end

to start.game
ball,
setpos [0 0]
setheading 75
setscore 10
everyone [clickon]
end

```

## Advanced ideas

Here is a long list of possible improvements, with some clues.

### ADD A SOUND EFFECT!

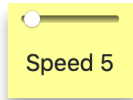
You can add a “very short *bong*” sound effect that plays when the ball bounces (doesn’t that give you a clue about where to insert the sound effect in your code?)

- Use a **wav** file, or record your own “toc” sound.
- Choose **Sound** in the **+** menu to open the **Import sound** dialog box.
- Once you have the sound icon on the page, click on it to test it.
- Right-click on the icon to open its dialog box, and set its name to a short, one-word name (like “**toc**”). While the dialog box is open, you can also take the time to make the icon invisible. It will still work.
- Add the name of the sound icon (in this case **toc** is a command that plays the sound) in the **bounce** procedure.

# Project 6 - Pong!

## MAKE SPEED A VARIABLE

Look at your procedures. Maybe you can figure out where you can change the speed of the ball, right? How about creating a slider and using its value for forward? Choose **Slider** in the **+** menu.



Right-click on the slider and name it **speed**.

```
1 to move.ball
2 ball,
3 forever [forward speed wait 1]
4 end
```

How about increasing the speed when the score reaches 15?

## SPICE UP THE GAME WITH RANDOM BOUNCES

The ball can bounce randomly on one of the walls. Instead of just using the same **bounce** procedure everywhere, you can have one **bounce.red** procedure that sets the heading to **-45 + random 90**. Use that special **bounce.red** procedure in the colour detection, for red.

## CEILING AND FLOOR

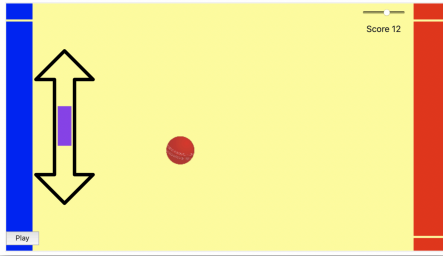
You can draw walls at the top and bottom of the page, and add more lines to your colour detection procedure:

```
to Ball_oncolour :prevColour :newColour
if :newColour = 'red' [hit]
if :newColour = 'blue' [miss]
if :newColour = ...
if :newColour = ...
end
```

You will need, however, a new type of **bouncing instruction**. The bouncing instruction for the red and blue walls will not do. You can try, but you will soon realize that it doesn't look natural. Explore bouncing methods such as

```
setheading heading - 90
setheading heading + 90
```

## VERTICAL MOVEMENT OF THE PADDLE



Right now, the paddle follows the mouse pointer everywhere on the page. How about if it glides only vertically, and stays on the left of the page?

In the `move.paddle` procedure, instead of using `forever [setpos mousepos]`

use

```
forever [sety last mousepos]
```

`Mousepos` reports a list of two numbers, an **x** coordinate, and a **y** coordinate. `Last mousepos` reports the **last** item of that list, which is just the **y coordinate**. Now the paddle will only move up and down.

Don't forget to freeze the turtle so the player can't cheat: Open the paddle's dialog box and check the ☒ **Frozen** box. Do the same for the ball.

## STOP THE GAME!

A nice addition would be a STOP button, in case a player wants to end the game without finishing it. Write the procedure below and In the **+** menu, choose **Button**.

```
to stop.now
stopall
end
```

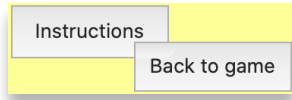
Give it this label: **I'M DONE**. and set its instruction to `stop.now`.

Name	button3
Label	<input type="text" value="I'M DONE"/>
On click	<input type="text" value="stop.now"/>

# Project 6 - Pong!

## INSTRUCTION PAGE

Create a second page, with some decor, and a text box with instructions for the player. Make sure you add a navigation button to go to the instruction page, and return to the game.



## Curriculum Links for Ontario

**C3.1** - Solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions.

**C3.2** - Read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code.





# Appendix A - Top 50 Lynx Primitives

This appendix contains a list of the most popular LYNX primitives - much more than what you need to create the projects in this book. If you wish to see a complete list of about 200 LYNX primitives, find the PDF called **List of LYNX Primitives**, on the LYNX web site ([lynxcoding.club](http://lynxcoding.club)), click on **Help** and scroll right in the LYNX **Resource Materials** section.

You can also check the online Help pages (from inside the LYNX editor), where the definitions and examples are a bit more elaborate.

All the examples below assume you have a **turtle** in the Work Area with its pen down, and in many cases, an empty **text box**.

When you see *number*, it means the Primitive requires an input (a number).

In the left column you have the primitive name and the short form (if any) and examples of use. In the right column you have an explanation of what the primitive does.

## Turtles and graphics

### TURTLE MOVEMENT

<b>FORWARD <i>number</i></b> <b>FD <i>number</i></b>  forward 100 cg repeat 4 [forward 100 right 90] cg repeat 36 [forward 100 back 80 right 10]	This command makes the turtle move in the direction it is facing. If the pen is down, the turtle leaves a line.
<b>BACK <i>number</i></b> <b>BK <i>number</i></b>  back 100 cg repeat 4 [back 100 right 90] cg repeat 36 [forward 100 back 80 right 10]	This command makes the turtle move backwards. If the pen is down, the turtle leaves a line.
<b>RIGHT <i>number</i></b> <b>RT <i>number</i></b>  repeat 4 [forward 100 right 90] cg right 90 (turns right 90 degrees, based on the previous heading)	This command makes the turtle turn to the right, without moving.  <b>Right</b> is relative to the current heading. If you absolutely want to point East, regardless of your current heading, see <b>setheading</b> below.

<p><b>LEFT</b> <i>number</i> <b>LT</b> <i>number</i></p> <p>repeat 4 [forward 100 left 90] cg left 90 (turns left 90 degrees, based on the previous heading)</p>	<p>This command makes the turtle turn to the left, without moving.</p> <p><b>Left</b> is relative to the current heading. If you absolutely want to point West, regardless of your current heading, see <b>setheading</b> below.</p>
<p><b>HOME</b></p> <p>pd right 11 forward 125 home</p>	<p>This command moves the current turtle to the centre of the page, its "home" base. The turtle's position is [0 0] and its heading is 0. If the pen is down, a line will appear between the current position and home position.</p>
<p><b>GLIDE</b></p> <p>glide 500 1 glide 500 5</p>	<p>This command makes the current turtle move forward. The first number is for distance and the second number is for speed. The effect of <b>glide 500 1</b> is very similar to <b>repeat 500 [forward 1 wait 1]</b>.</p>
<p><b>POS</b></p> <p>home show pos right 45 forward 250 show pos</p>	<p>This reporter returns the current position of the current turtle, as a list of two numbers. Use this reporter to find out the position of a turtle, so you can later use this value with <b>setpos</b> in a <b>setup</b> procedure. The position at the centre of the page is [0 0].</p>
<p><b>SETHEADING</b> <i>number</i> <b>SETH</b> <i>number</i></p> <p>home setheading 90 right 180 setheading 90</p>	<p>This command sets the heading of the current turtle. The heading corresponds to the values on a compass: 0 (or 360) is due North, 90 is due East, 180 is due South and 270 is due West. <b>Setheading</b> is <b>absolute</b>: if you run the instruction <b>setheading 90</b> twice, the heading will always be due East. <b>Right</b> and <b>left</b> are <b>relative</b>. If you run the instruction <b>right 90</b> several times, the turtle will keep turning, always 90 degrees relative to its current heading.</p>

# Appendix A - Top 50 Lynx Primitives

<p><b>SETPOS</b> <i>[xcor ycor]</i></p> <p>setpos [50 50] setpos [-50 50]</p>	<p>This command moves the turtle to the position indicated. If the turtle's pen is down, it leaves a line. The position at the centre of the page is [0 0].</p>
<p><b>SETSHAPE</b> <i>clipart_number_or_name</i> <b>SETSHAPE</b> <i>list_of_numbers_or_names</i></p> <p>setshape 1 setshape 'cat' setshape [1 2] repeat 50 [forward 3 wait 6]</p>	<p>This command sets the shape (clipart) of the current turtle. You must first bring at least one clipart into your clipart pane. Right-click on a clipart if you want to give it a name.</p> <p>The input of setshape can be a single number or clipart name, but you can also use a list of numbers or names. If you do so, the turtle will switch shape each time it moves (forward, back, setx, sety, setpos, etc.)</p>
<p><b>SETX</b> <i>number</i></p> <p>setx 100 setx -100</p>	<p>This command moves the turtle horizontally to the <b>x</b> coordinate indicated.</p>
<p><b>XCOR</b></p> <p>home show xcor right 45 forward 100 show xcor</p>	<p>This reporter returns the <b>x</b> coordinate of the current turtle.</p>
<p><b>SETY</b> <i>number</i></p> <p>sety 100 sety -100</p>	<p>This command moves the turtle vertically to the <b>y</b> coordinate indicated.</p>
<p><b>YCOR</b></p> <p>home show ycor right 45 forward 100 show ycor</p>	<p>This reporter returns the <b>y</b> coordinate of the current turtle.</p>

## TURTLE STATE

<b>ST</b>  repeat 10 [st wait 5 ht wait 5] st	Stands for Show Turtle. This command makes the current turtle visible. If the turtle is already visible, this command has no effect.
<b>HT</b>  repeat 10 [st wait 5 ht wait 5]	Stands for Hide turtle. This command makes the current turtle invisible. If the turtle was already invisible, this command has no effect.
<b>SETSIZE <i>number</i></b>  setsize 20 setsize 40 repeat 5 [setsize 20 wait 5 setsize 40 wait 5]	This command sets the size of the turtle. The original size is <b>40</b> . The minimum size for visibility is <b>5</b> , and the maximum size is <b>160</b> .

## TURTLE DRAWINGS

<b>PENDOWN</b> <b>PD</b>  pendown forward 150	This command puts the pen down for the current turtle. The turtle will draw a line as it moves.
<b>PENUP</b> <b>PU</b>  pendown forward 100 penup forward 100	This command lifts the pen for the current turtle. The turtle won't draw a line as it moves.
<b>PENERASE</b> <b>PE</b>  pendown forward 125 penerase back 100	This command sets the pen eraser for the current turtle. The turtle erases as it moves.
<b>CLEAN</b>  pd forward 150 wait 20 clean	This command removes all the graphics on the current page, without moving the turtle(s). The graphics that are "cleaned" include the turtle graphics and the stamped turtles, including a large "background" stamped turtle.

# Appendix A - Top 50 Lynx Primitives

<b>CLEARGRAPHICS</b> <b>CG</b>  right 11 forward 9999 cg forward 9999 clean	This command cleans all the graphics on the current page and moves the current turtle to the centre of the page ( <b>home</b> ). The graphics that are “cleaned” include turtle graphics (lines) and stamped turtles, including a large “background” stamped turtle.
<b>SETPENSIZE</b> <i>number</i>  setpensize 5 pd forward 100 setpensize 20 forward 100	This command sets the turtle's pen size which determines the thickness of the lines it will draw. The original pen size is <b>1</b> . The maximum pen size is <b>99</b> .
<b>SETCOLOUR</b> <i>number_or_colour_name</i> <b>SETC</b> <i>number_or_colour_name</i>  setcolour 15 setcolour 'blue' repeat 10 [setc colour + 1 wait 5]	This command sets the colour of the turtle and it's pen. If the turtle has its original shape, it changes colour to show the pen colour. The input can be the name of a colour or a number. See <b>Appendix C - LYNX colour chart</b> .

## Text primitives

<b>PRINT</b> <i>word_or_list</i> <b>PR</b> <i>word_or_list</i>  print 'hello' print 'there' print [hello there]	This command prints a word or a list in the current text box, at the position of the cursor (insertion point). The <i>word_or_list</i> is printed in the text box and the cursor drops to the next line. Use <b>insert</b> if you want the cursor to stay on the current line.
<b>INSERT</b> <i>word_or_list</i>  insert 'hi' insert ' there, ' ; there is a space before the "t" and after the comma wait 20 insert [how are you?]	This command inserts a word or a list in the current text box, at the position of the cursor (insertion point). The <i>word_or_list</i> is printed in the text box and the cursor stays on that line. Use <b>print</b> if you want the cursor to move to the next line after printing.

<p><b>CLEARTEXT</b> <b>CT</b></p> <p>Type something in two different text boxes text1, cleartext</p>	<p>This command clears the text in the current text box. The current text box is the last text box that you have created, or the last one that you have addressed with a comma (ex.: text1,).</p>
<p><b>ANNOUNCE word_or_list</b></p> <p>announce 'hello' announce [hello there] question [Your name is?] Type your name in the dialog box and click OK announce sentence [Good day] answer</p>	<p>This command displays a word or a list in an alert box. Any action taking place in your project is stopped while the alert box is showing.</p>
<p><b>QUESTION word_or_list</b></p> <p>question [Your name is?] Type your name in the dialog box and click OK announce sentence [Good day] answer</p> <p>question [Your name is?] Type KIM in the dialog box click OK if answer = 'KIM' [say sentence 'Hi' answer]</p>	<p>This primitive displays a dialog box containing the question (<b>word_or_list</b>). After the question is answered, use the primitive <b>answer</b> to process what was typed in the dialog box.</p>
<p><b>ANSWER</b></p> <p>See the example for <b>question</b> above.</p>	<p>This reporter returns the last answer that was typed in a <b>question</b> dialog box as a word. The value returned by this primitive remains unchanged until a new <b>question</b> is displayed and answered. If you close the <b>question</b> dialog box using the <b>Cancel</b> button, <b>answer</b> returns an empty list. Any action taking place in your project is stopped while the dialog box is showing.</p>
<p><b>SHOW word_or_list</b></p> <p>show 'Hello' show [Hello there] (LYNX displays Hello there without the brackets) show xcor show pos</p>	<p>This command prints the <b>word_or_list</b> in the Command Centre. The quotation marks and square brackets will not be printed. This command is useful when you want to print information without having to create a text box.</p>

# Appendix A - Top 50 Lynx Primitives

## *SAY word\_or\_list*

say 'Hello'  
say [This is it!]

Type something in a text box, then type this in the Command Centre:

say text1

This command makes your computer speak the

*word\_or\_list*.

This is a nice way to get the computer to say aloud what you have typed in a text box.

## Conditionals and other primitives

**IF**  
*condition*  
*list\_of\_instructions*

home  
if heading = 0 [ht wait 5 st]

If instructions are often used to stop recursive procedures. Create this procedure:

to spiral :length  
if :length > 100 [stop]  
forward :length  
right 90  
spiral :length + 10  
end

Now type this in the Command Centre:

spiral 0

Without the *if* instruction, this spiral would grow forever.

This command or reporter runs the condition. If the condition reports

*true*, it executes the

*list\_of\_instructions*. If the condition is *false*, nothing happens.



<p><b>IFELSE</b> <i>condition</i>  <i>list_to_run_if_true</i>  <i>list_to_run_if_false</i></p> <p>(Put a turtle on the page and two shapes in the clipart pane)  ifelse heading = 0 [setshape 1]  [setshape 2]  The turtle takes shape 1 if it is heading North; otherwise it takes shape 2.</p> <p>to getage  ifelse score = 10  [page2]  [announce 'keep playing']  end</p> <p>If the value in the text box named score is zero, LYNX displays the page2. Otherwise, it displays an alert box.</p>	<p>This command or reporter runs the condition.</p> <p>If the condition reports <b>true</b>, it executes the <i>list_to_run_if_true</i> (the instructions in the 1st set of brackets).</p> <p>If the condition reports <b>false</b>, it executes the <i>list_to_run_if_false</i> (the instructions in the 2nd set of brackets)</p>
<p><b>REPEAT</b> <i>number</i>  <i>list_of_instructions</i></p> <p>repeat 4 [forward 125 right 90 wait 5]  repeat 5 [ht wait 5 st wait 5]  repeat 10 [setsize 20 wait 2 setsize 40 wait 2]  cleartext  repeat 5 [print 'hi' wait 10]</p>	<p>This command runs the <i>list_of_instructions</i> the number of times indicated.</p> <p>The instructions inside the square brackets will be repeated the number of times specified.</p>
<p><b>WAIT</b> <i>number</i></p> <p>home right 90  repeat 4 [forward 10 wait 10]  repeat 10 [setsize 20 wait 2 setsize 40 wait 2]  cleartext  repeat 5 [print 'hello' cleartext wait 10]</p>	<p>This command creates a pause in the execution of instructions. The duration is in tenths of a second:  <b>wait 10</b> means wait <b>1</b> second.  <b>wait 5</b> means wait <b>1/2</b> second</p>

# Appendix A - Top 50 Lynx Primitives

## FOREVER *list\_of\_instructions*

Create the **Move** procedure then type this in the Command Centre:

**forever [move].**

```
5 ▾ to move
6   forward random 100
7   right random 360
8   wait 10
9   end
```

This command runs the *list\_of\_instructions* inside the square brackets repeatedly... forever. Use the **Stopall** icon, to the immediate left of the Command Centre, to stop the action, or use the command **stopall**.

## STOPALL

Create a procedure such as the one below. Create a button that runs this procedure.

My project

Procedures

```
1 ▾ to halt
2   stopall
3   end
```

Name button1

Label Stop it!

Click halt

Type this in the Command Centre.

**forever [forward 1 wait 1]**

Click on the button **Stop it!**

This command stops all running procedures and processes, including turtles and buttons. You can create a procedure that uses **stopall** and then place that procedure inside a button or turtle. You can also type **stopall** in the Command Centre.

## EVERYONE

*list\_of\_instructions*

Put two or more turtles on a page then type this in the Command Centre:

**everyone [forward 50]**

They seem to move at the same time, but...

**everyone [forward 50 wait 10]**

Now they move clearly one after the other.

**everyone [setsize 75 wait 10]**

This command makes every turtle on the page run the *list\_of\_instructions* one at a time, until all the turtles have done it.

If you want them to move concurrently (at the same time) use the **talkto (tto)** command.

**tto [t1 t2] forward 50 wait 10**

<pre> ASK turtle_name list_of_instructions ASK list_of turtle_names list_of_instructions ASK text_box_name list_of_instructions ASK list_of text_box_names list_of_instructions  ask 't1' [forward 50] ask [t1 t2] [forward 50 wait 10] Notice that t2 moves only AFTER t1 has completed the forward and the wait.  t1, forward 50 (t1 does it) forward 50 (t1 is still listening) ask 't2' [forward 50] (now t2 does it) forward 50 (but if you don't use "ask", t1 is still listening and will go forward 50) </pre>	<p>This command asks the turtle (or the turtles) to run the <i>list_of_instructions</i>. If you ask a <i>list_of_turtles</i> to run the instructions, the first turtle in the list will run the instructions, then the next turtle will do the same only after the first turtle is done, and so on.</p> <p>This command does not change who is the current turtle. The current turtle is the last turtle that you created, or the last turtle that you clicked on, or the last turtle that you addressed using the comma (ex.: t2,).</p> <p>The same rules apply to text boxes (see the examples below). Also used with <i>everyone</i> and <i>talkto</i>.</p>
<pre> TALKTO turtle_or_list_of_turtles TTO turtle_or_list_of_turtles TALKTO text_box_name TTO text_box_name  talkto 't1' forward 100 talkto [t1 t2] forward 100 t2, back 100 </pre>	<p>This command makes the turtle(s) or text box current. This is the only way of making many turtles do the same thing at the same time.</p> <p>You must use the <i>talkto</i> method when you wish to talk to (address) more than one turtle.</p> <p>For a single turtle or text box, you can also use the <i>comma</i> feature: the name of a turtle or a text box followed by a comma is equivalent to a <i>talkto</i> instruction. The <i>comma</i> method works for one turtle or one text box.</p>

## Appendix A - Top 50 Lynx Primitives

### CLICKON

Put 2 turtles on the page with their pen down. Space them apart. Create the **Circle** procedure and put the procedure in the On Click instruction inside both turtles.

```
1 to circle
2 repeat 36 [fd 10 rt 10]
3 end
```

Name	t1
Xcor	0
Click	circle

Then type this in the Command Centre:

**everyone [clickon]**

This command simulates a click on the current turtle.

If a turtle is programmed to react to a mouse click, using the command **clickon** is the same as actually clicking on that turtle.

### CLICKOFF

Click on the turtle and wait for a while as it wanders around. Then type this in the Command Centre:

**clickoff**

```
1 to wander
2 forever [move]
3 end
4
5 to move
6 forward random 100
7 right random 360
8 wait 10
9 end
```

Name	t1
Xcor	-121
Click	wander

This command simulates an “unclick” on the current turtle.

If a turtle is programmed to react to a mouse click and is currently running its “on click” action, **clickoff** is the same as actually clicking off that turtle.

If the turtle is not running its “click” action, then **clickoff** does nothing.

<p><b>RANDOM number</b></p> <p>Make sure you have a turtle and text box on the page.</p> <p>right random 360 forward random 100 setcolour random 140 setsize random 120 print random 6</p>	<p>This reporter returns a random non-negative integer (including 0) that is less than the number indicated.</p>
<p><b>PICK word_or_list</b></p> <p>show pick 'Hello' show pick [Kim Sam Leo]</p> <p>Assuming you have three turtles on the page: ask pick [t1 t2 t3] [forward 50]</p>	<p>This reporter returns one random character of a word, or one random item of a list.</p>
<p><b>NOTE number duration</b></p> <p>note 65 10</p>	<p>Plays a musical note. The first input is the pitch (1 to 127) and the second input is the duration, in tenths of a second (1 to 255). See Appendix G for more information.</p>
<p><b>REST duration</b></p> <p>note 65 10 rest 10 note 67 10</p>	<p>Inserts a silent interval in a series of notes.</p>
<p><b>SETVOLUME sound-name number</b></p> <p>setvolume 'mysound' 10 mysound</p>	<p>Sets the volume of the specified sound at the level selected from 0 to 100. The first input is the name of a sound that you have imported in your project.</p>
<p><b>VOLUME sound-name</b></p> <p>show volume 'mysound' 50</p> <p>to louder setvolume 'mysound' volume 'mysound' + 25 end</p>	<p>Reports the volume of the specified sound. The input is the name of a sound that you have imported in your project.</p>

## Appendix B - Common coding errors

### SPELLING MISTAKES!

LYNX understands **forward**, **right** and **setsize** for example.

LYNX will give you an Error Message if you type **forwrdr**, **rite** and **stesize**, for example.

**Always check your spelling if you get an error message.**

### SPACES IN OBJECT NAMES

When naming turtles, text boxes, sliders and pages be sure you **don't** use spaces before, inside, or after the name. If a text box is called **"My text"** with a space between MY and TEXT, the instructions **my text**, **setmy text** and **say my text** will not work:

```
I don't know how to my
```

If a text box is called **"Mytext"**, the instructions **mytext**, **setmytext** and **say mytext** will work

If a text box is called **"My\_text"**, the instructions **my\_text**, **setmy\_text** and **say my\_text** will work

### SPACES IN PROCEDURE NAMES

When creating procedures, be sure to use single words, with **no spaces**. If you created a procedure called **"big square"**, you won't be able to run it. LYNX will display an error message:

```
I don't know how to big
```

**My.square**, **my\_square** and **MySquare** are all names that work.

### PROCEDURES THAT DON'T WORK

There are many rules to follow when creating procedures.

```
to square
```

```
; This draws a square
```

```
repeat 4 [fd 50 rt 90]
```

```
end
```

THE WORD **TO**, A SPACE, AND ONE WORD

COMMENTS START WITH A SEMICOLON **“;”**

INSTRUCTIONS, AS MANY AS YOU WANT

THE WORD **END**, BY ITSELF ON THE LAST LINE.

Also, remember that procedure names must be new to LYNX. You cannot use a LYNX primitive (built-in vocabulary) or other procedures you have already created for this project.

If you forget the word **end** at the end of a procedure, all the procedures that follow that procedure will not work.

## WRONG TYPE OF QUOTATION MARK

Sometimes, when copying and pasting code from other sources, you may end up with typographical (curly or slanted or French) quotation marks such as these: `'yellow'` or `"yellow` or `«yellow`. These will **not** work. Make sure you use straight quotation marks, like `'yellow'` or `"yellow`. If you get the wrong type of quotation mark, just erase and retype the quote in the LYNX editor.

## NOT TALKING TO THE RIGHT TURTLE OR TEXT BOX

When you run a turtle command, the turtle that will execute it is the current turtle. The current turtle is:

- a) the last turtle that you have created, or
- b) the last turtle that you have clicked on, or
- c) the last turtle that you called using the “comma” method (`paddle,`) or the **talkto** command.

If a procedure is meant to be executed by a specific turtle, simply state its name followed by a comma (no space) at the beginning:

```
to bounce  
ball,  
setheading...  
end
```

The same rules apply to text boxes.

## BUTTON NOT WORKING

Did you forget to “link” the button to a procedure? If yes, do a right-click on the button and, in the On Click field, choose the procedure you want to run when the button is clicked.

If you ever change the name of the procedure, the button will stop working. In that case, reopen the button’s dialog box and relink it to the actual procedure, found in the **On click** drop down menu.

## Appendix B - Common coding errors

### NOT PAYING ATTENTION TO ERROR MESSAGES

Error messages are important. Extremely important. They always contain a clue about what the problem is, and *generally, on which line the error is*, in the case of procedures.

---

**I don't know how to annnounce in winner on line 43**

You tried to execute a word that is not a LYNX primitive nor a procedure you have created. Sometimes this is just due to a typo (**annnounce** instead of **announce**). Sometimes it is because you forgot a typographical element (**print hello** instead of **print 'hello'** - in the first case **'hello'** is just a word to be printed. In the second case, **hello** looks like a command to execute).

---

**I don't know what to do with...**

Some primitives “report” a value. In these cases, you have to tell LYNX what to do with the value. **Heading**, for example, reports the heading of the current turtle. If you execute just the word **heading**, you get an error message. If you execute **show heading**, or **print heading**, or **setheading heading + 90**, the value reported by **heading** will be used by **show**, **print**, or **setheading** - this is fine.

---

**No ... found for ...**

You tried to execute a turtle command and there is no turtle on the page. Or a text command, and there is no text box on the page.

---

**... doesn't like ... as input**

Certain primitives like only certain types of inputs. **Print** can print anything (**print 'hello' print 44**). **Setheading** likes only numbers. **Forward** likes numbers not greater than 9999.



## UNWANTED SPACE

So many times in this book, we told you “one word, no space”. This is true for the names of pages, turtles, text boxes, sliders, sounds... That also means that there are no spaces when you use these words.

```
show text1
t1,
ask 't1' [forward 50]
setslider1 slider1 + 10
```

These are all good. There is no space in `text1`.

Remember: `text1` and the other examples above are the names of objects, **no space needed**.

`forward 50` is a primitive with an input, **it needs a space**.

## Appendix C - Lynx colour chart

LYNX turtles can be any colour. Colours in LYNX are numbered like this:

0	10	20	30	40	50	60	70	80	90	100	110	120	130
1	11	21	31	41	51	61	71	81	91	101	111	121	131
2	12	22	32	42	52	62	72	82	92	102	112	122	132
3	13	23	33	43	53	63	73	83	93	103	113	123	133
4	14	24	34	44	54	64	74	84	94	104	114	124	134
5	15	25	35	45	55	65	75	85	95	105	115	125	135
6	16	26	36	46	56	66	76	86	96	106	116	126	136
7	17	27	37	47	57	67	77	87	97	107	117	127	137
8	18	28	38	48	58	68	78	88	98	108	118	128	138
9	19	29	39	49	59	69	79	89	99	109	119	129	139

To set a turtle or background to a particular colour, run the **setcolour** (**setc** for short) or **setbg** command, followed by the chosen colour number:

**setc 127** or **setbg 'green'**



BLACK 9



WHITE 0



GREY 5



RED 15



ORANGE 25



BROWN 35



YELLOW 45



GREEN 55



LIME 65



TURQUOISE 75



CYAN 85



SKY 95



BLUE 105



VIOLET 115



MAGENTA 125



PINK 135

Colour numbers go in tens. Each set of ten contains the shades of a particular colour. For example, shades of yellow are from 40 to 49 and shades of orange — from 20 to 29. The smaller the number in a set of ten, the lighter the shade, the bigger the number — the darker the shade.

16 colours are considered “basic”. They have not only numbers, but also names. These colours and their names are shown on the left. For these colours you can type **setc colour\_name** in the Command Centre instead of just **setc colour\_number**

For example:

**setc 'violet'** and **setc 115**  
do the same thing.

If you want your turtle to be one shade darker than the VIOLET 115 colour, you need to run an instruction such as:

```
setc 116 No name is available for that shade.
```

In short: 110 to 119 is the range of violet shades, 115 is the middle point and corresponds to:

```
setc 'violet'
```

In that set of ten, anything above 115 is a darker shade, anything below 115 is a lighter shade.

And the same is true for all the other sets of shades.

When to use names and when to use numbers? Names only exist for 16 basic shades. Using names makes your code more “human”. But when you do calculations, use numbers, like this:

```
repeat 140 [setcolour colour + 1 stamp forward 20]
```

## Appendix D - Finding things

### Free Resources on the web

At the time of printing of this book (September 2020) the sites listed below were *up-to-date* and had free resources. We cannot guarantee this will still be true when you read this coding book. Please verify copyright rules *before* using resources from these sites.

#### SITES TO FIND CLIPART WITH TRANSPARENCY

PNG and JPEG are 2 file formats that work with LYNX. Take a tour of these two sites:

<http://pixabay.com> and <https://www.freepngimg.com>

Search their databases for clipart with a transparent surrounding.

Search a term (ex.: car), pick one, click on Choose a size. 200 x 200 is generally sufficient for a turtle to use as a shape.

See next page about the importance of transparency for PNG files.

#### SITE TO FIND BACKGROUND PHOTOS

<https://unsplash.com>

Search their database for background photos (wallpaper type).

Some images may be quite large, but LYNX resizes the image to the project size when you import it.

#### SITES TO RECORD YOUR VOICE

Your computer or phone may already have a voice recording tool. If not, here are two options:

<https://vocaroo.com> or <https://www.audacityteam.org/>

Save your recording in WAV or MP3 file format and don't make the recording too long.

## SITES TO FIND SOUND EFFECTS

BBC Sound Effects has a searchable database of WAV files.

<http://bbcsfx.acropolis.org.uk>

Click on **Duration** to sort by length. Use **short** audio clips with LYNX.

A second option is below. Read the Attribution rules carefully.

<http://soundbible.com>

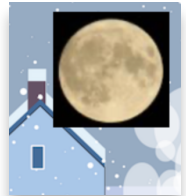
## Web site where you can edit your images

At the site below there is a free editor that will allow you to modify images. Be aware that there is a free mode, with ads, and also a paid version.

<https://pixlr.com/>

## About transparency of PNG files

JPG files, like photos, simply do not support transparency. The same applies to screen captures, even if you capture it in PNG. These are always **rectangular objects**. For example, here is a picture of the moon, screen captured on a black background used as LYNX clipart and turtle shape. The image on the right is what you will see - probably not what you want.



In order to get a nice “cut out” image of the moon, or any object, you have two options. Either you:

- Edit the screen capture or jpg file to **erase** the surrounding (can be easy or difficult), and **save it as PNG with transparency**, or
- Search for true PNG images with transparency. They often have a grey or checkered surrounding indicating the transparent area, like this image of a tree.



You need to **download** the file to your computer or your personal work space in the cloud (Google Drive for example), and **import** it to an empty clipart box. Click anywhere in the empty box and a “+” sign appears.

## Appendix E - FAQ

### Using LYNX on Chromebook

When using a Chromebook computer, we assume that all your LYNX files are in the LYNX cloud (this is the same for any LYNX user), and all the external files that you will try to import into LYNX (clipart, sounds, voice recordings, screen captures) *will come from your Google drive*. Screen captures and other files *may* be in your local **Downloads** folder, but for the sake of simplicity, we assume that you have brought these into your Google drive as well. In any case, you should check both **Google Drive** and **Downloads** for the clipart, sounds and screen captures you want to use.

Your **Google Drive** is in the left column of any Browser window or Open dialog box.

### Using LYNX on iPad

When using LYNX on an iPad, be aware that a “click” in any instruction you read in this book, becomes a “touch” on an iPad. Also, since there is no “right-click” on an iPad, you must use the **Project tree** to open the dialog box of a turtle, a text box, a button, etc. Click on the **Project tree** icon as shown here, click on the **small triangle** to expand a page and see its contents, click on an object for example a Turtle or Button, and finally click on **Edit** to open its dialog box, or **Delete** to permanently delete the object.



Screen captures are much harder to do, so we suggest that you find pre-made clipart or create your clipart on a regular computer.

### How to do a screen capture to use as a Preview image or as a clipart

As mentioned above, screen capture will produce rectangular clipart, with *no* transparent surrounding. You have to edit the screen capture to erase the surrounding yourself and save the image as a PNG file.

## IN WINDOWS:

Launch **Snipping Tool** in the **Windows Accessories**. In **Mode**, choose **Rectangular Snip**. Use the **Rectangular** mode. Select a region on your screen, that is tight around the image you wish to copy. Copy (**Ctrl+C**) the resulting clipart from **Snipping Tool**. Now click in an empty spot in the LYNX Clipart Pane and paste (**Ctrl+V**).

If you want to use the image as a Preview for your project, you first have to save it as a PNG file, then navigate to it when selecting the Preview image.



## ON A MAC:

**Pre-Mojave, macOS 10.13 or lower:** Press **Command-Shift-4**. You get a crosshair pointer. Select a region on your screen, that is tight around the image you wish to copy. The image is saved on your desktop. Double-click on that file to open it in **Preview**, and copy it from there. Click in an empty spot in your Clipart Pane and paste the clipboard (**Command-V**).

**Mojave, Catalina, macOS 10.14 or higher:** Press **Command-Shift-5**. Select **Clipboard** in the **Options**. Select a region on your screen, that is tight around the image you wish to use and copy it (**Command-C**). Now click in an empty spot in your LYNX Clipart Pane and paste the clipboard (**Command-V**).

You can save your screen capture on your computer, open it in **Preview** and use the tool **Instant Alpha** to determine which colour should be transparent. It is better, for that, to make screen captures on a solid white background.

## ON A CHROMEBOOK:

Press **Ctrl-Shift-SwitchWindows** () simultaneously and select the region you wish to capture. This will create a file in the **My Files> Downloads** folder. Then use the  method to load the image as clipart in the Clipart Pane or in **Project Properties**, choose **Select a Preview image**

## Appendix F - Sharing

There are many reasons to share your projects. In this book, there is an interactive card (Thank you essential workers, Mother's day, Happy Birthday...), secret code and flashy art projects and two games! As a proud coder, you might just want to share all projects that you create.

LYNX makes sharing easy. You can share a project with **some friends** of your choosing (text message, email), or anybody who follows you on Facebook or Twitter, or with **every LYNX user in Canada**.

To read more about project management and sharing, see the documents **Sharing your projects**: go to the [www.lynxcoding.club](http://www.lynxcoding.club), click on **Help**, then on the tab **Sharing a Project** in the section **All About LYNX**.

### Before sharing

#### PREPARE YOUR PROJECT

You must name your project and save it before it can be shared. You should be on Page1 (or the starting page of your project) when you save your project. Very important: Do NOT include in your project any personal info that can identify you or say where you live. No photo of yourself or full name.

Make sure your project is usable in **Play mode**. You may need

- a button or a clickable turtle to **start** the action, if any;
- a button to **stop** the action, if you use a **forever** action without a stop rule;
- a text box with instructions, if needed. The text box can be visible at all times, or only appear when you click an "Instructions" button (use the commands **showtext** and **hidetext**), or if there are lots of instructions, the **Instructions button** could lead to an **Instructions page**;

```
1 ▾ to show.instruct
2   text1, showtext
3   end
4
5 ▾ to hide.instruct
6   text1, hidetext
7   end
```

Show instructions  
Hide instructions

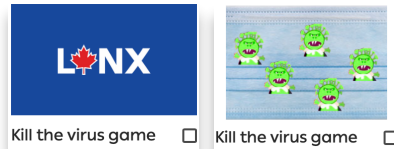
```
8 ▾ to instruct.page
9   page2
10  end
```

- a **startup** procedure that has some instructions to reset things in your project, and open a specific page. See this feature in *Project 3 - Interactive Thank You card*.



## CREATE A PREVIEW IMAGE

Creating a **Preview image** is always a good idea, even when you keep your project private. When your project has a **Preview image**, it is recognizable in your list of projects in the LYNX cloud. Here's your project without and with a **Preview image**:



Creating a **Preview image** for your project is not mandatory for sharing your project. *However, it is highly recommended*, as it attracts more viewers when you share it. Before you can select a **Preview image** for your project, you must first create the image file. For example, save a screen capture of the nicest page in your project. See *Appendix E - FAQ* for more information about making screen captures.

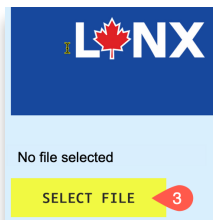
Now go to **My Projects**, where your projects reside in the LYNX cloud. Click on your project once (1), to open it in **Player mode**.



In **Player mode**, click on **Properties** (2).



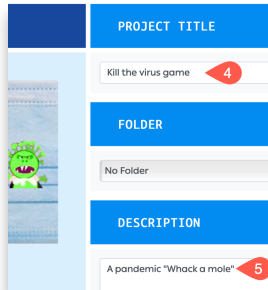
On the **Properties** page, click on **Select File** (3), under the default preview image:



Navigate to the file that you have prepared for the **Preview image**. It will **not** appear immediately on the **Properties** page, but it will appear when you click on **Save** in a moment.

## Appendix F - Sharing

Type a **Title** (4), and maybe include some instructions in the **Desc** (description) field (5) or write some info about the project.



The screenshot shows a project form with three main sections: PROJECT TITLE, FOLDER, and DESCRIPTION. The PROJECT TITLE section has a text input field containing "Kill the virus game" with a red callout bubble labeled "4" pointing to it. The FOLDER section has a text input field containing "No Folder". The DESCRIPTION section has a text input field containing "A pandemic 'Whack a mole'" with a red callout bubble labeled "5" pointing to it.

Click on **Save** when you are done. Your project is now recognizable on the page **My Projects**.

You are all set for sharing now.

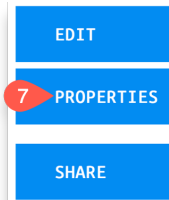


### Start sharing

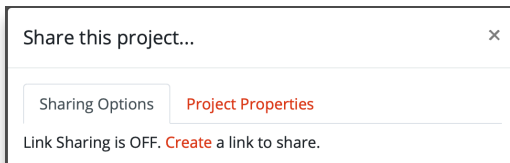
There are two places to launch this sharing process: from **within the LYNX editor**: click on this icon (6) at the top-left of the page...



or look at your project in **Play mode** (click on your project, on the **My Projects** page), and click on **Share** (7):



Both methods will open the **Sharing** dialog box.

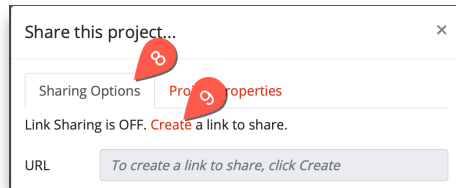


The screenshot shows a dialog box titled "Share this project..." with a close button (X) in the top right corner. Inside the dialog, there are two tabs: "Sharing Options" and "Project Properties". The "Project Properties" tab is selected. Below the tabs, it says "Link Sharing is OFF. Create a link to share."

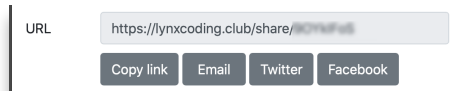
## Sharing with some people you know

If your project does not have a **Preview image**, a **Title** and a **Description**, you can click on the **Properties** tab now to do that, following the instructions in the section just above.

If you have done all this, stay on the **Sharing Options** tab (8) and click on **Create** (9). LYNX will create a URL that represents your project in the LYNX cloud.



The four buttons just below the URL allow you to:



- **COPY LINK:** Choose this option so you can paste the link anywhere, for example in an e-mail or in a messaging application on your computer.
- **E-MAIL:** This button opens a new e-mail message, with the link already in the body of the message. Just add the other details and send!
- **TWITTER / FACEBOOK:** Remember to choose the friends you would like to share your project with otherwise everyone will see your project. Your friends will see the **Preview** of your project and then click / press on it to be taken to lynxcoding.club

## Appendix F - Sharing

### Sharing with all (private vs public)

By default, the projects that you create and save are **Private**. A private project only appears in your **My Projects** page. **Only you can see and modify this project**, unless you share it.

If you uncheck the box labelled **Private** in the **Project properties**, the project becomes **public**. It will also appear on the **All Projects** page. When a project is **public**, other members of the LYNX community will be able to see it, make changes to its code and other content and use it as a starting point for their own projects. They will be able to make changes to your original project, but they can only save the modified project as their own, in their own LYNX cloud. *They cannot change your personal project even if it appears in a public space.*

#### IN SHORT

**KEEPING YOUR PROJECT PRIVATE:** Nobody will see or edit your project, but if you choose to share it by e-mail, text message or social media, people specifically selected by you will see your project in **Play mode**. They will not see the LYNX editor, the tools, the Command Centre and the Procedure and Clipart Panes.

If you *uncheck* Private in the Sharing dialog box, then the following is also true:

**MAKING YOUR PROJECT PUBLIC (UNCHECK PRIVATE):** The project will appear on the **All Projects** page. Anyone visiting the LYNX home page will see it, and will be able to make changes to it. If they have a LYNX account, they will be able to save the modified project as their own. YOUR project **will stay exactly the way you made it**. If they don't have a LYNX account, they will still be able to see, open and edit your project, but they won't be able to save the modified project.

#### HOW TO EMBED YOUR PROJECT IN A WEBSITE OR BLOG

You can also embed your project in a Web site or blog. Copy the code provided in the field **Embed on your site**, in the lower part of the **Sharing** dialog box.

#### MORE INFO ABOUT SHARING INCLUDING HOW TO EMBED

To read more about sharing, see the three documents on the LYNX web site: click on **Help > All About LYNX > Sharing a Project**.

## Appendix G - Musical Notes

LYNX can play musical notes; it uses the standard MIDI numbering system where 60 is the middle C.

The command **note** plays a note. The first input is a note number (pitch) and the second is the duration in tenths of a second.

The command **rest** creates a pause. Its input is a duration, in tenths of a second.

The image on the right shows the values for the pitch. However, LYNX goes beyond the piano keyboard and the note number can be any number between 1 and 127. The value for the duration can be any number between 1 and 255.

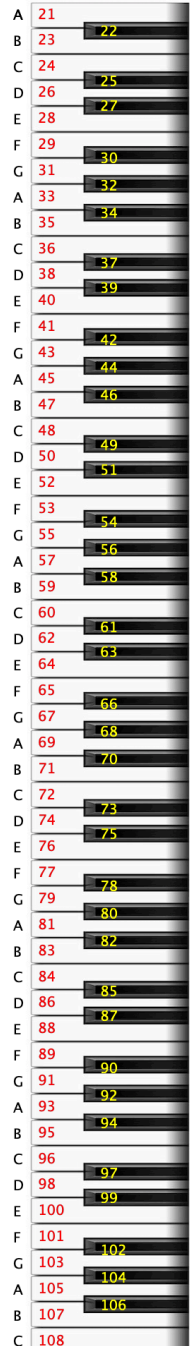
For example:

```
note 60 10
note 5 10
note 120 30
```

```
to mysong
note 70 5
note 68 3
note 66 3
rest 2
note 61 3
note 63 3
note 61 3
note 63 3
note 61 5
end
```

**Important:** If you are using a note or a series of notes in your code, the note or the notes will play in full before anything else happens. If you want to play a note or a set of notes by itself (as an independent, parallel process), and continue with the rest of your program, use **launch** to play the notes:

```
launch [note 67 10 rest 5 note 60 30]
launch [mysong]
```



## ADDITIONAL RESOURCES

On the Home page of [lynxcoding.club](http://lynxcoding.club), click on **Help** on the top bar. Then click on any tab in the sections ***All about LYNX, Projects & Lessons with LYNX*** or ***LYNX Resource Materials***.

Also, there are wonderful resources, including short **How To** videos and long recorded webinars at:  
[sites.google.com/view/lynxcoding-org](https://sites.google.com/view/lynxcoding-org)



## What is LYNX?

LYNX is a designed-in-Canada coding app from a team with decades of experience making coding apps for K-12 students. All the core designers have worked with the late Dr. Seymour Papert of the MIT Media Lab, the father of educational computing.

Lynx is a cloud-based programming environment for learners to create sophisticated projects. This text-based language supports computational thinking without the strict rules of other coding languages, such as JavaScript.

Think of LYNX as the next step after using block-based coding tools!

You can easily share your LYNX Projects with your friends.

*Thanks to a Federal Government CanCode grant from ISED Canada, the Lynx coding app is free to use by Canadian residents.*

---

*We have successfully integrated LYNX into our culturally responsive mathematics project in a number of Grade 3 to Grade 8 classrooms in Ontario. The dynamic nature of LYNX allowed students to explore the different interpretations of mathematical concepts inherent in the work, which then allowed us to more fully analyze students' mathematical understanding. The students loved writing code and in doing so they had numerous opportunities for problem solving.*

*Ruth Beatty, PhD  
Faculty of Education, Lakehead University*



---

Not sure if this book is for you or your child? You can watch a two-minute movie called *The Missing Link* at [lynxcoding.club](https://lynxcoding.club) which gives you an idea of what you can do with Lynx.

Ready to give it a try? Everything you need is here: [LYNXCODING.CLUB](https://lynxcoding.club)

---

*Text based coding is the way to go. With block coding, you're just doing, there's no need to understand the why behind it all. Text based coding makes students critical thinkers and problem solvers and it definitely makes us more aware of the English language.*

*Lisa Martin, Teacher, La Scie NL*

---

## What you need to run LYNX:

An Internet connection and a Windows computer, a Mac or a Chromebook. Lynx will run on iPads and Android tablets, though the tablet screen size may be challenging—and a keyboard is recommended!