

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).

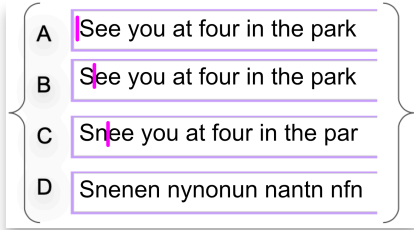
WHAT IS PSEUDO-CODE?

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 (**message**) 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
```

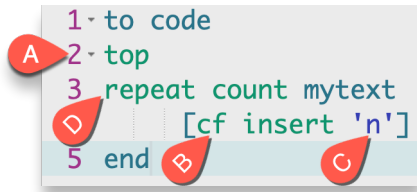
```
THIS IS WHAT LYNX RETURNS FOR OUR EXAMPLE
```

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]

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



The instruction `repeat count mytext` is the same as `repeat 27` times (to match the number of characters in the 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.

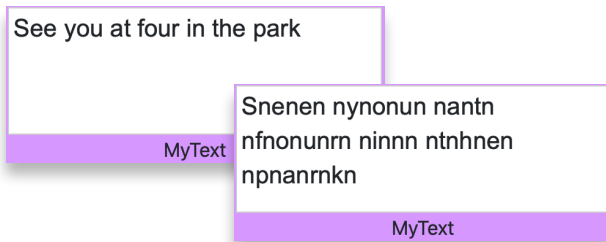
Click on the **+** menu then choose **Button**.

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 nnonwn?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.



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

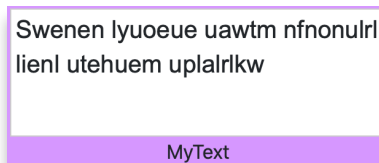
13 ▾ to better.code
14 ▾ top
15 repeat count mytext
16   [cf insert 'n']
17 end
```

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

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

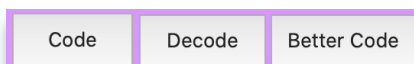
13 ▾ to better.code
14 ▾ top
15 repeat count mytext
16   [cf insert pick 'mwelun']
17 end
```

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:



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.



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?

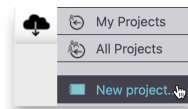
Mission Impossible: Next level Secret Codes

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.

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.

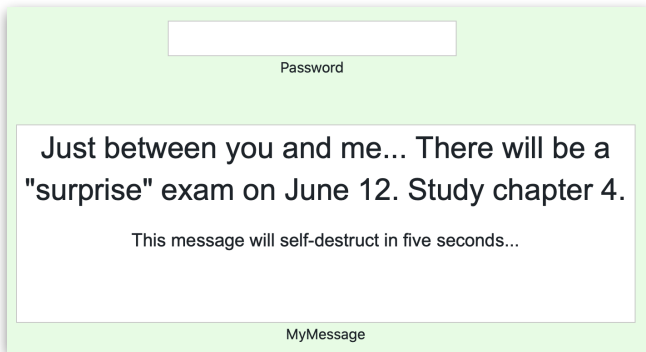


If you wish, make a nice background (use **setbg** to get a coloured background).

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 what 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.

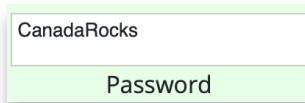
Then type this in the Command Centre:

```
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 PASSWORD IS THE NAME OF THE TEXT BOX  
CanadaRocks IT REPORTS THE CONTENTS OF THE TEXT BOX
```

And now, type this:

```
show password = 'canadarocks' IS THIS EQUAL TO THE  
true CONTENTS OF THE TEXT BOX?
```

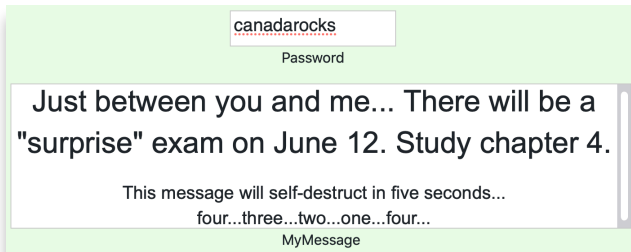
Lynx reports **true** because the contents of the text box (what is reported by **password**) is really equal to **'canadarocks'**. Casing (uppercase, lowercase) does not have to match.

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

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

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

The text box should reappear:

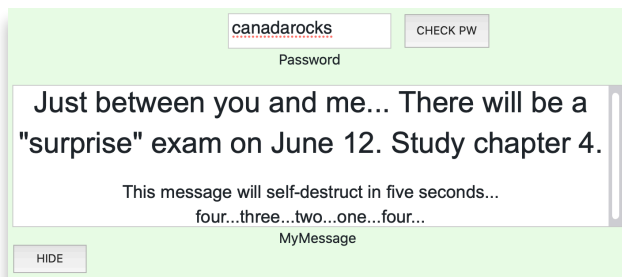


OK, you have all the pieces now. Let's code!

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 all 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.



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, edit its dialog box and 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!*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.

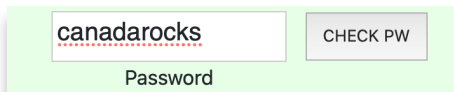
Last but not least, code a procedure to validate the password.

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:



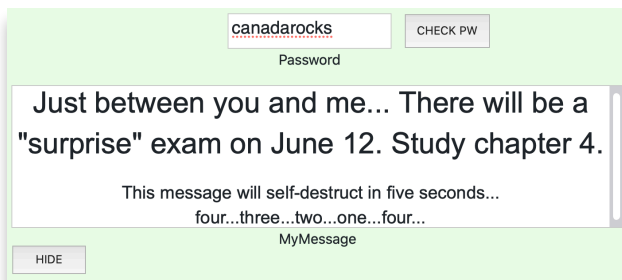
canadarocks

CHECK PW

Password

TEST, AND SHARE

Do you have all these elements?



canadarocks

CHECK PW

Password

Just between you and me... There will be a "surprise" exam on June 12. Study chapter 4.

This message will self-destruct in five seconds...
four...three...two...one...four...

MyMessage

HIDE

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 one character to the right and delete two characters
repeat count mytext [insert pick 'wmunel' cf cf]	Delete every third 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.