Day 2: Balancing and Graphing in Snap!

Infusing Computing Professional Development 2019

Welcome!

Activities for Today's Coding Session

Activity 1: Balancing Scales

Use Snap*!* to explore the Law of the Lever and factors affecting balance

Activity 2: Graphing

Explore points and slope using the Cartesian Coordinate system in Snap*!*.

Goals for Today's Coding Activities!

- **Get comfortable with reading code** and fixing bugs! Sometimes when your students program, they will make mistakes, which we call bugs! Today you will experience reading through and fixing code.
- The first activity will help you **learn how two Sprites can interact**. Remember from yesterday that a Sprite is the character that you program. Today we will use the touching block and the broadcast block to show how they can interact!

- The second activity will help you understand **lists** and how they can be used to store data. The activity also showcases how **Parsons Problems** can be used to build equations.

What is Computational Thinking

Computational Thinking (CT) is a problem solving process.

CT is essential for developing programs, but it can also be used to support problem solving across all disciplines, including the humanities, math, and science.

In this PD we will highlight 4 essential elements of CT:

Remember to Pair Program

During the coding sessions, we expect you to switch who is **driving** (writing coding) and who is **navigating** (reading the instructions and reviewing the code) with your partner so you each get a chance to program and have the benefit of working with someone.

Designate your computers: one as the **Driving Machine** and the other as the **Navigator**.

The slides will tell you when to switch (it is based on task)

At this time, **choose who will be Teacher A and who will be Teacher B** before moving on.

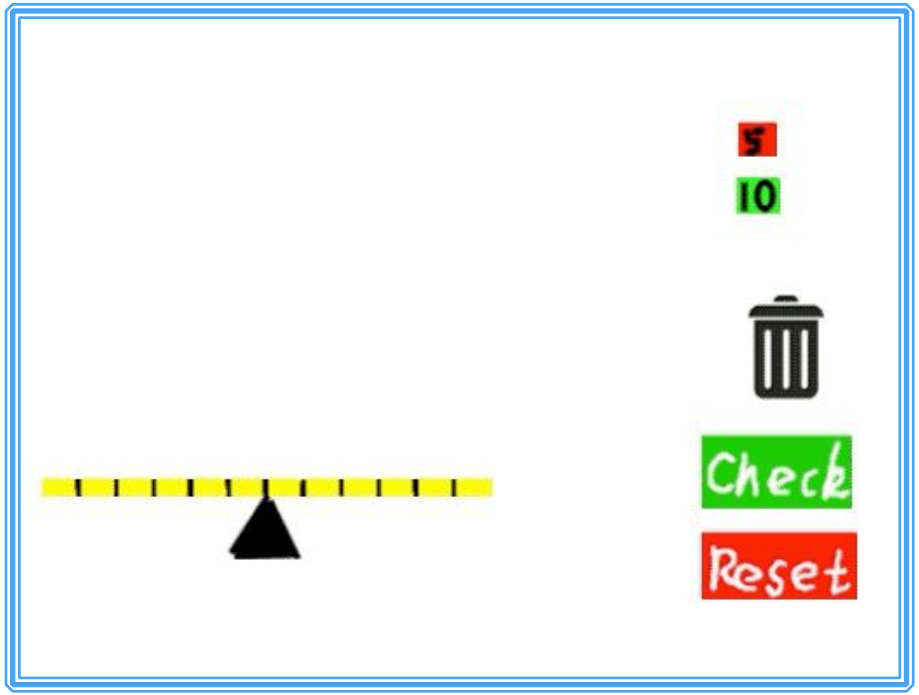
Pair Programming!

Teacher A: Drive!
Teacher B: Navigate!

Activity 1: Balancing Scales

Goals & Project Demo

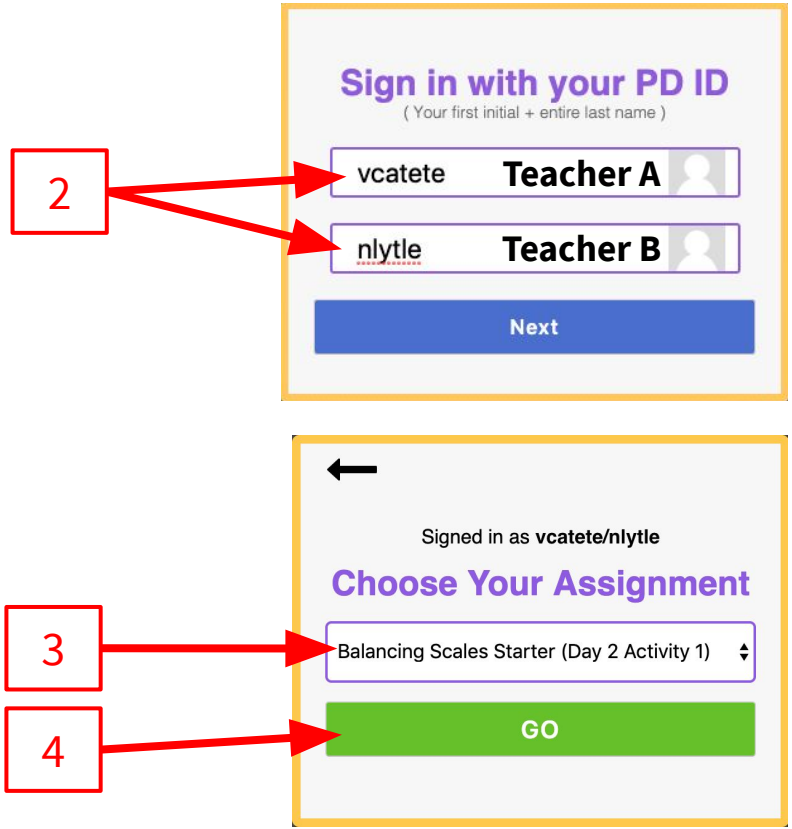
- Play through the Balancing Activity
- Fix the Scale's code
- Fix the Weight's code
- Add "say" blocks to tell Check's code to give feedback
- Create new weights!



Sign in to Snap!

1. Go to stemc.csc.ncsu.edu/2019_pd_snap to "Sign in with PD ID"
2. Type the PD ID of Teacher A & Teacher B and click "Next"
3. Choose "**Balancing Scales Starter**" from the "Choose Your Assignment" window
4. Click "Go"

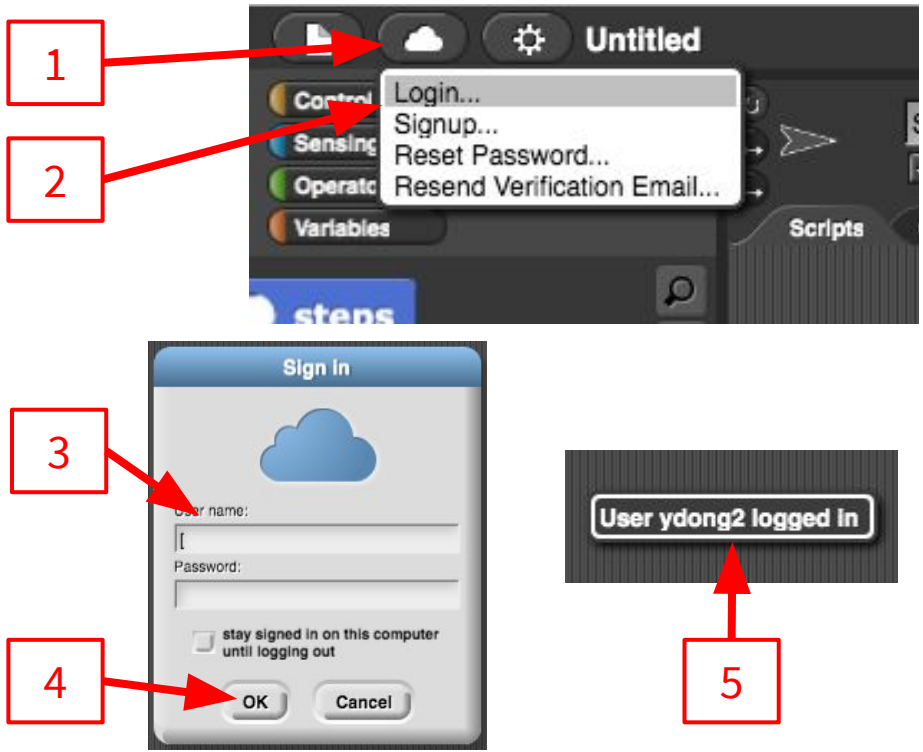
(Need help finding your PD ID? Click here)



Login to Snap Cloud

1. Click on the "**Cloud**" Icon
2. Select "**Login...**"
3. Input **Teacher A**'s login information in the pop up window
4. Click on **Okay** button
5. Look for confirmation information

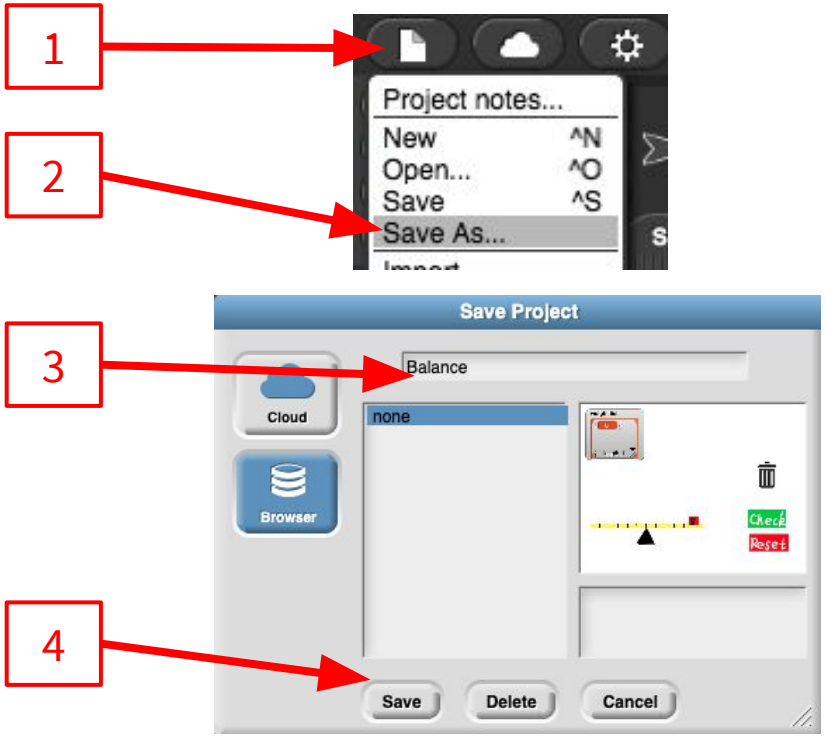
(If neither partner has a Snap! account, click here)



Save Your Code Before Start

It's a good practice to Save your code every time you work on something!

1. Click the **File Icon**
2. Select "**Save as...**"
3. Type in "**Balance**"
4. Click on the **Save** button



Walk through the Balancing Scales

In today's examples you are provided with **Starter Code**.

Starter code is helpful to provide scaffolding for students and to focus their efforts on aspects of the program that are important to the learning goals.

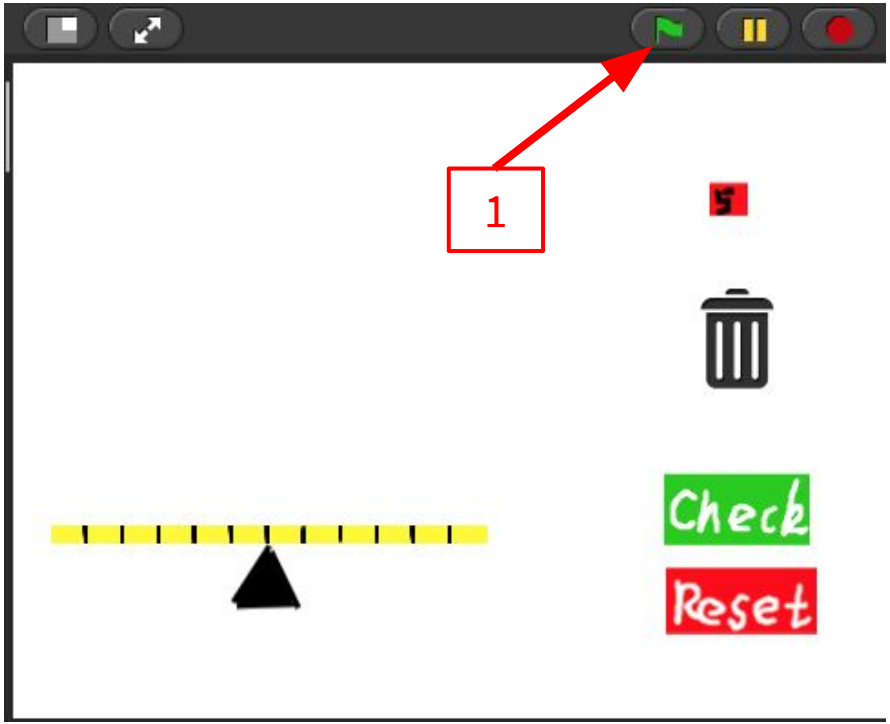
In a typical assignment, our development team usually provides starter code that controls basic program functionality (timing, tutorial text, etc.).

Depending on the time available in class and students' prior coding exposure, we may adjust how much starter code is provided.

Walk through the Balancing Scales

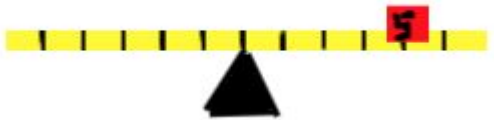
Let's play through the **starter project** and see what it does now.

1. Click on the Green Flag to run the program.



Walk through the Balancing Scales

There are **five actions** that you can do in the environment:



1. Click on the red 5lb **weight** (5) - it should **make a copy**!
2. Drag a **weight** to the **scale** and drop it!
3. Click on the Check to see the scale **adjust** for the weight.
   a. It seems that the scale **leans** in the **opposite direction**!
4. Drag the red weight to the trash - the weight should **disappear**...
   a. The **weight didn't disappear**!
5. Click Reset to reset the stage

There are bugs in the starter code! We have some work to do!

Bug

A bug refers to a mistake in the code that leads to an incorrect behavior

Walk through: Buggy Code

The first activity we're doing is called a "**buggy code**" activity.

Buggy code problems encourage students to think about how the program should actually work and tasks them to explain why the current version is incorrect.

The questioning process goes as follows:

What usually happens when …
What does the program do?
Is this correct? Why or Why not?
Try to find where the error is in the code
See if you can fix it

Remember Yesterday?

There was buggy code in Mrs. Hill's guessing game logic.

Step 1: Fix the Scale

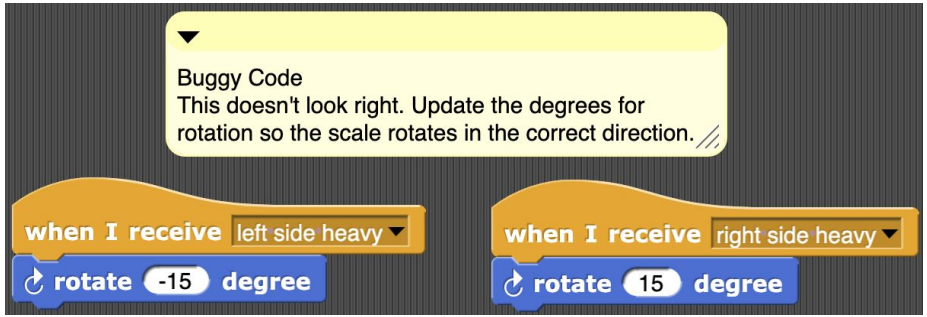
Let's fix the Scale so it tilts the correct direction.

1. Click on the Scale sprite
2. At the bottom of the Scale's scripts, there is a comment called Buggy Code.
3. Change the number of degrees the scale rotates so that the scale rotates in the correct direction.

Step 1: Fix the Scale

How does your code compare?

1. Click on the Green Flag and play the game again.
2. Put weight on each side of the scale and click on Check



Does the scale tilt to the correct side?

Decomposition

Notice the two 'when I receive Message' blocks **decompose** the tilting behavior of the scale into two categories.

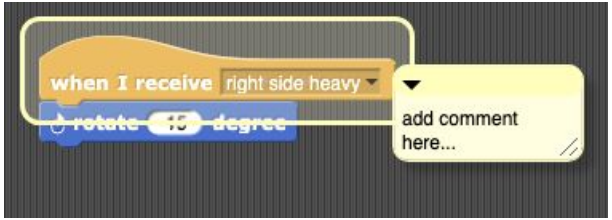
Step 1: Fix the Scale

Now, let's place a comment near the blocks to explain what they do.

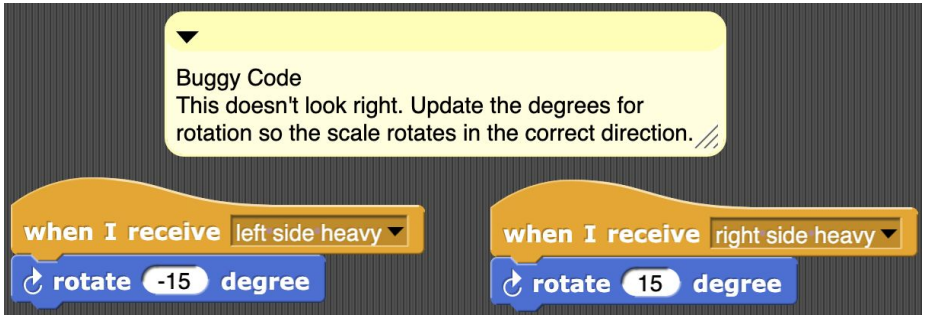
1. Right-Click in the coding workspace to add a new comment.

2. You can also drag your comment towards the code blocks to attach it to a specific segment.

1



2



Save your code!

Switch for the next activity!

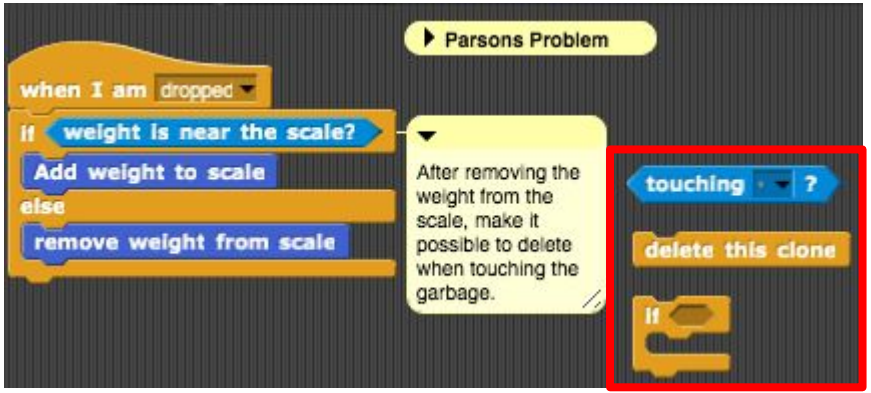
Teacher A: Navigate!
Teacher B: Drive!

Step 2: Complete the Weight Code

Let's look at why the weight won't disappear when dropped on the garbage bin.

1. Click on **weight** sprite.

It seems that some code blocks are detached from the script.



Step 2: Complete the Weight Code

This type of coding activity is called a **Parson's problem**.

> Parson's Problems **provides** students with each of the **specific blocks** that they will be using at the beginning. These blocks are **out of order** and not connected.

Parson's Problems can **relieve extraneous cognitive load** by reducing the students need to hunt for blocks in an unfamiliar environment.

We also use Parson's Problems to **provide scaffolding** for students.

> Using the provided blocks like Legos, they can assemble them in various ways until they reach a working solution.

Parson's Problem
It's easier to assemble a bird house using a kit rather than looking for pieces in a lumber yard.

Step 2: Complete the Weight Code

The 

block is used to create and report an instance (a clone) of any sprite.

This is a temporary clone and behaves like it's parent sprite.

Once the game or animation is over, you don’t need the clones any more and they go away.

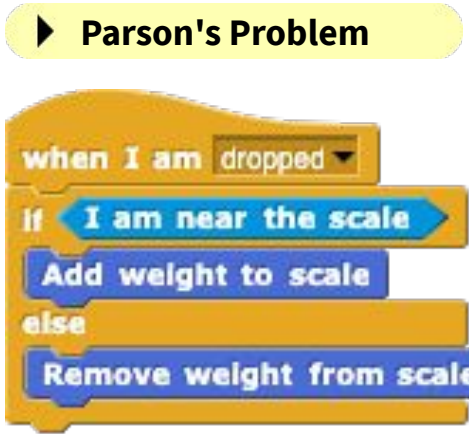
When you click on the weight block, it creates a clone.

Step 2: Complete the Weight Code

Using clones, here are logical steps to make a weight disappear:

After removing a weight from scale
If the weight touches the garbage
    Delete this clone

1. Figure out **where to put the three code blocks** to make the weight disappear when dropped on the garbage bin.

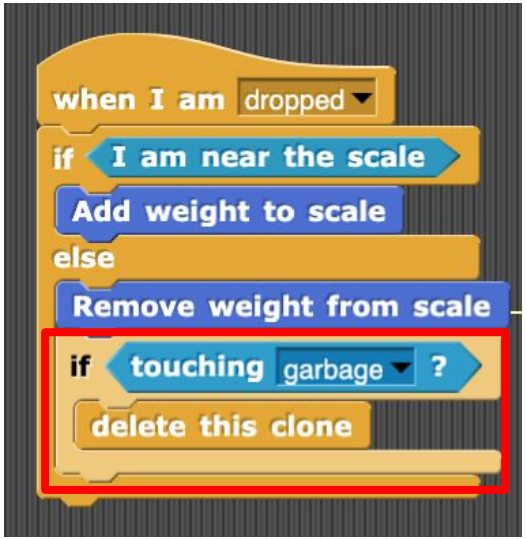


Check your answer

Step 2: Complete the Weight Code

How does your code compare?

Click the 5 to create a **new weight** and drop it on the **garbage bin**, does the new weight disappear?



Algorithms

You've just finish an **Algorithm** that handles what happens when a weight is dropped on the Stage!

Decomposition

This Algorithm **decomposes** how to handle when a weight is dropped and reacts accordingly.

Step 2: Complete the Weight Code

Abstractions

Notice how the circled blocks are all **custom blocks** that **abstract** away the **complexity** of their code and focus on the **important steps** with **meaningful names**.

This resembles how we describe **relevant and important steps** in a problem in real life. For example, when we describe how we open doors, we say if the door has a handle, we pull; if the door has a bar, we push. But we do not describe how our arms move when pulling or pushing the door. The arm actions are **Abstracted** away in the pull and push action.

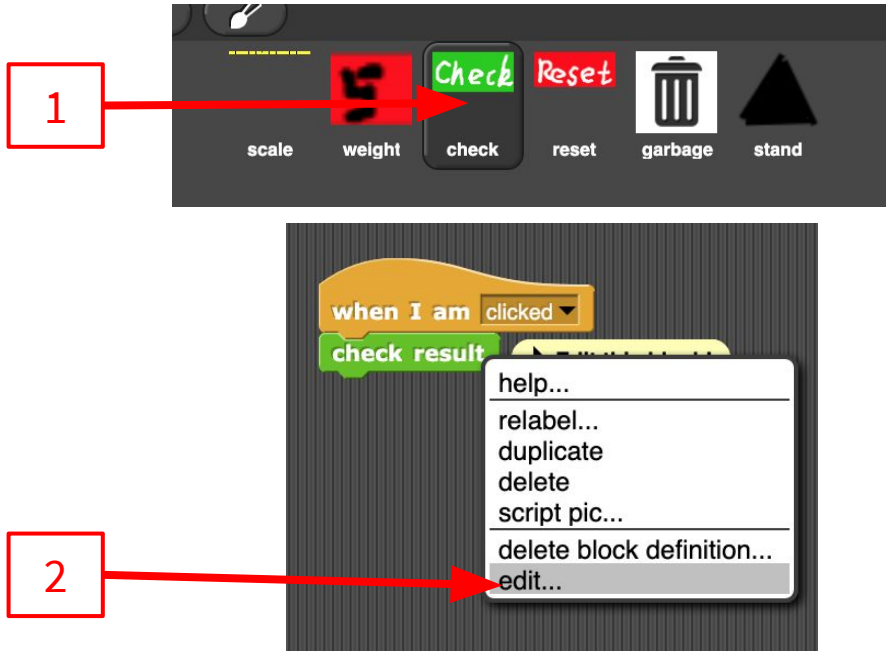
Save your code!

Step 3: Give Proper Feedback

Did you notice how the starter code does not give feedback when the check button is clicked?

We are going to add that!

1. Click on the **Check** Sprite.
2. Right click and edit the "check result" block



Abstractions

The **check result** custom block **abstracts** away the logic used to determine which side is heavier

Step 3: Give Proper Feedback

Incomplete code and worked examples often go hand in hand. As we often provide '**starter code**' for students, it is often incomplete and requires students to do some coding on their end.

For less experienced students, we provide examples with similar functionality and hope students can transfer that information to another location.

In other instances of incomplete code, we may provide pseudo code or "common language" descriptions to describe what needs to be added.

Pseudo code

We used pseudo code in the box on slide 22 to describe the algorithm for deleting a weight

Step 3: Give Proper Feedback

Let's look at the incomplete code in the check result custom block.

The nested if conditions determines which side of the scale is heavier.

If right side is greater than left side
    Broadcast right heavy message
Else
    If left side is greater than right side
        Broadcast left heavy message
    Else
        Broadcast equal heavy
        Say “They are equally heavy” for 4 secs

Pseudo code

Step 3: Give Proper Feedback

Do you see that if the blocks are equally heavy, the program says “They are equally heavy” for 4 seconds?

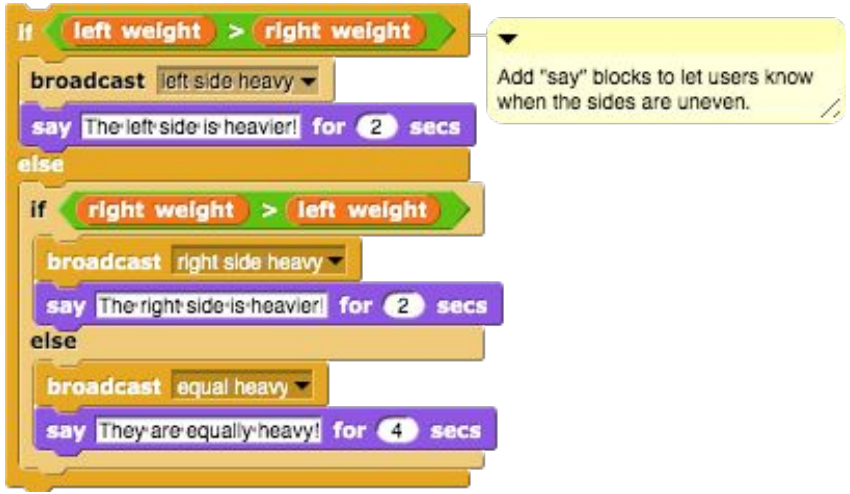
1. Add say _ for _ secs block in the if blocks to make the check button give the correct feedback on which side is heavier.

Check your answer

Step 3: Give Proper Feedback

Is this what your code looks like?

Put weight on each side of the scale and press the check button, does the check button give correct feedback?



Decomposition

Notice how the **nested if-else** blocks **decompose** the weight comparison into **three conditions**?

This **resembles** how we think when solving a multiple comparisons problem in real life.

Save your code!

Switch for the next activity!

Teacher A: Drive!
Teacher B: Navigate!

Step 4: Create a 10 lb Weight

In this activity we are going to duplicate a 5lb weight sprite and modify it to make a new 10lb weight.

This is an example of an **extension activity** where students can add in their own creations. In this example, it doesn't matter how much mass the new weight has. For CT, the goal is for the student to use pattern recognition and in a physics context, the goal is to have varying weights to stack and experiment with.

Students often have varying ideas for extensions. As a facilitator you don't need to know all the answers. You can practice asking students reflective and developing questions.

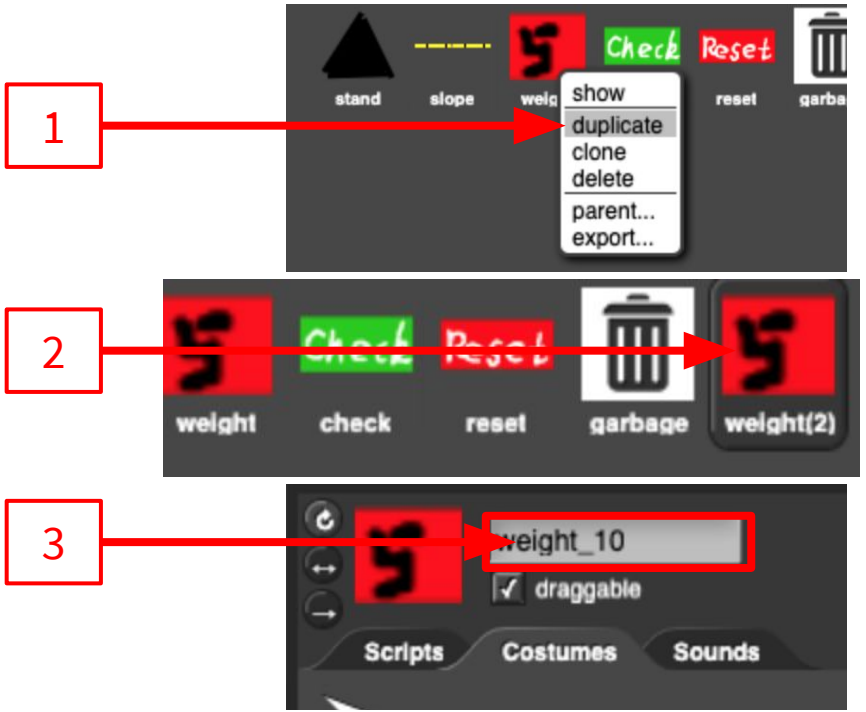
Extensions

Extensions allow students to test new concepts using creative techniques.

Step 4: Create a 10 lb Weight

Now, let's make the 10 lb weight Sprite by duplicating the 5 lb weight sprite!

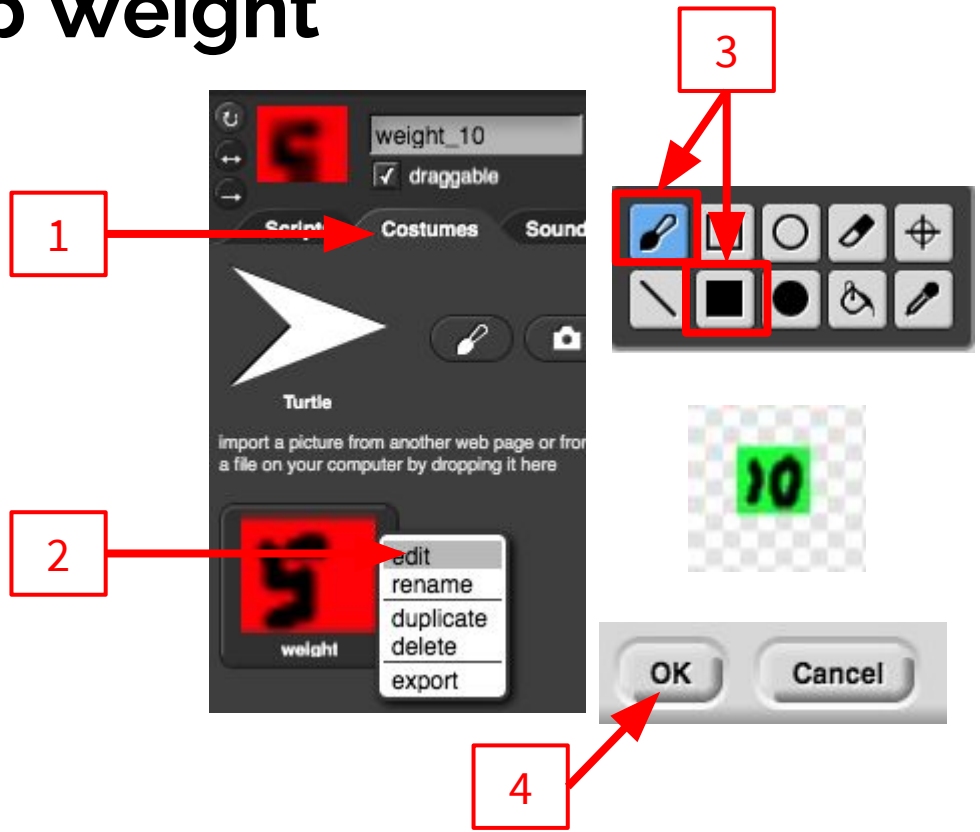
1. Right click on the 5 lb weight sprite and select duplicate
2. Click on the new weight(2) sprite to see its code
3. Rename the weight(2) sprite to weight_10



Step 4: Create a 10 lb Weight

Let's make a new costume for the 10 lb weight.

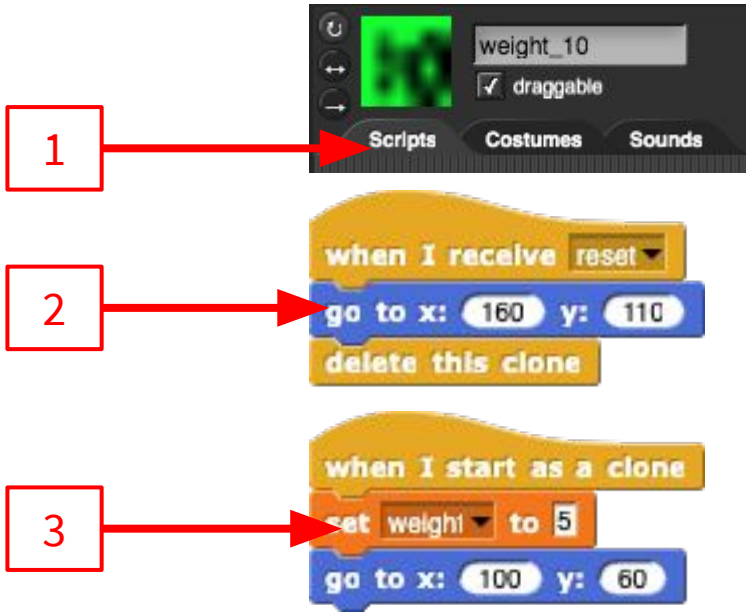
1. Click on the costumes tab to change the sprite's costume
2. Right click on the costume and select edit
3. Draw a new costume for the 10 lb weight with paint brush tool
4. Click okay when done



Step 4: Create a 10 lb Weight

Let's modify the code for the 10 lb weight sprite!

1. Switch back to the Scripts tab
2. Change the location of the 10 lb weight when it resets so that it does not hide the 5 lb weight
3. Set the weight variable to 10 so a new value can be considered when calculating scale balance



Check your answer

Step 4: Create a 10 lb Weight

Is your code similar?

If your code looks different but works for you, it's fine!

Now you can add more weight!

Pattern Recognition

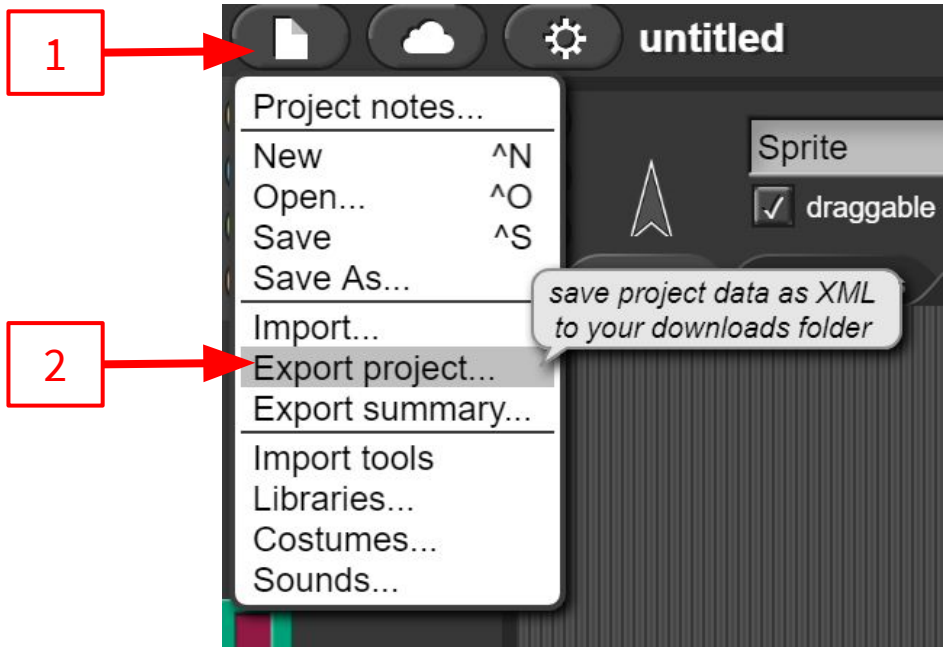
Notice how we recognized similarities and differences in the two weights and only modified the important parts to make a new weight cube!

Save your code!

You've Completed Activity 1!

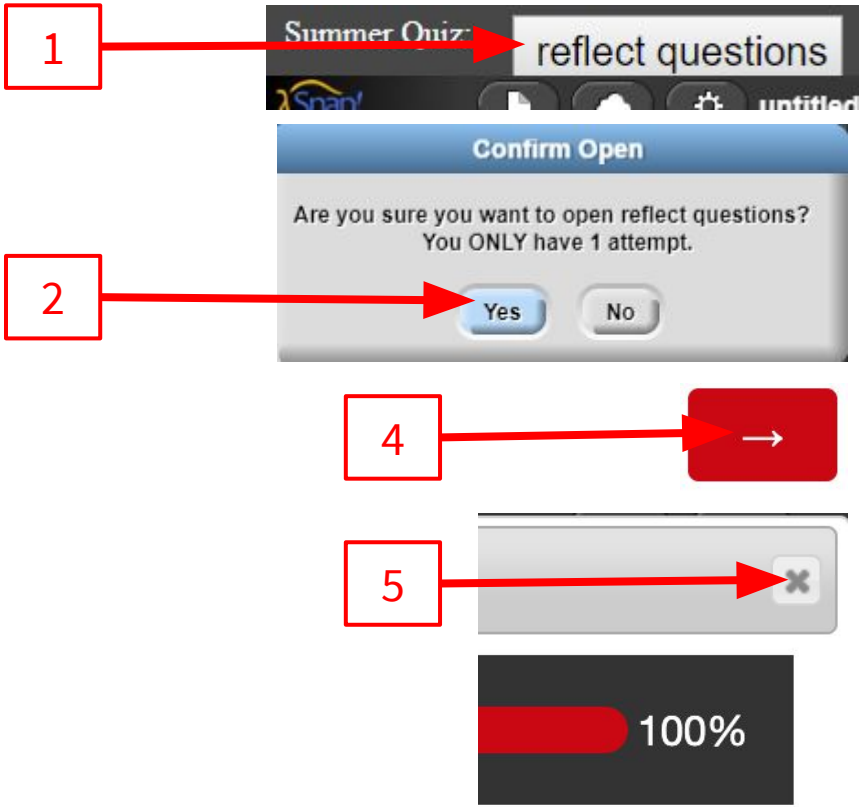
If you want to export and download the project to your computer, here is how:

1. Click on the File Menu
2. Select on Export project
3. Choose where you want to save the project file and click save



Let's Reflect!

1. Click "Reflect Questions" in the top right.
2. Press **Yes** to begin the reflection.
3. Use the **red arrow button** to go to the next question.
4. Finish out the reflection quiz.
5. After finishing the quiz, use the **cross button** on the top right corner of the survey window to close.



Switch for the next activity!

Teacher A: Navigate!
Teacher B: Drive!

Activity 2: Graphing Coordinates

Goals & Project Demo

- Play through the Graphing Coordinates Activity
- Broadcast activity:slope and wait
- Mark the coordinate points
- Move the grid to the right
- Draw the correct slope

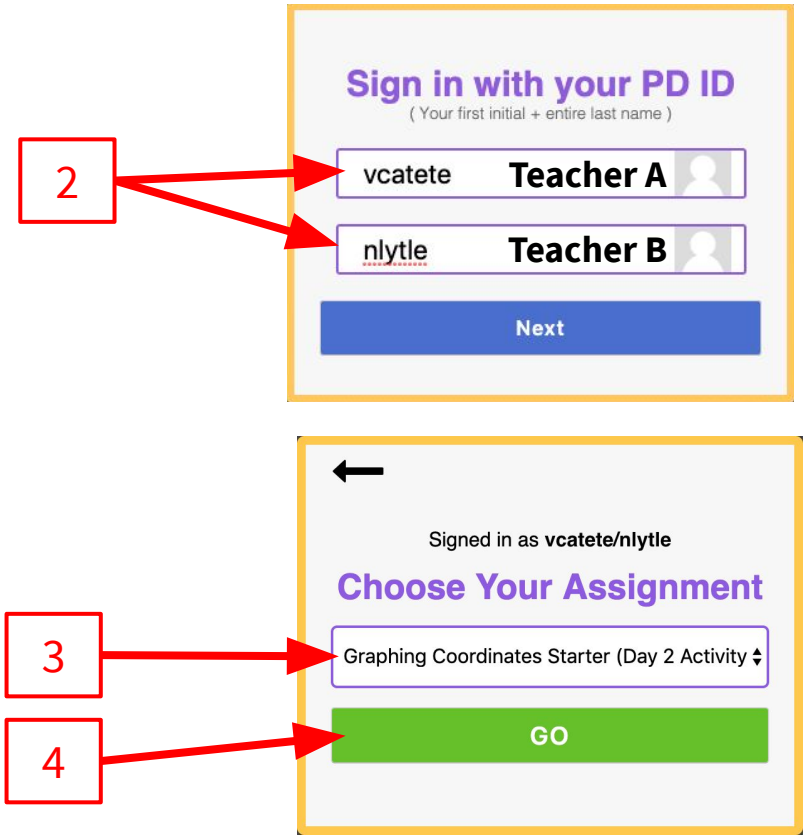
If there is time...

- Add to the Summarize Activity Coordinates
- Add to the Introduce Activity Slope and Summarize Activity Slope

Sign in to Snap!

1. Go to stemc.csc.ncsu.edu/2019_pd_snap to "Sign in with PD ID"
2. Type the PD ID of Teacher A & Teacher B and click "Next"
3. Choose "**Graphing Coordinates Starter**" from the "Choose Your Assignment" window
4. Click "Go"

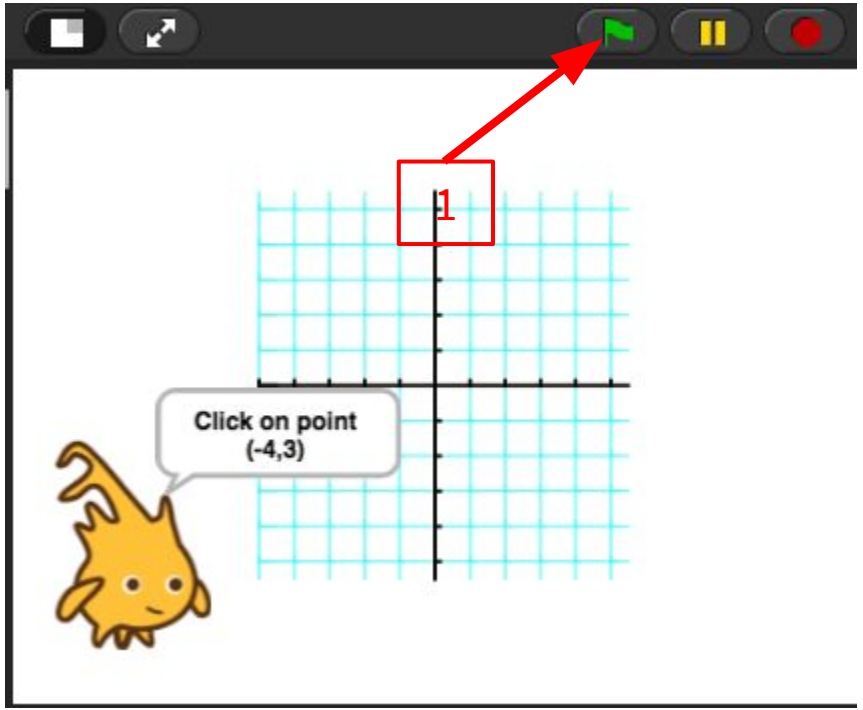
(Need help finding your PD ID? Click here)



Walk through the Balancing Scales

Let's play through the **starter project** and see what it does now.

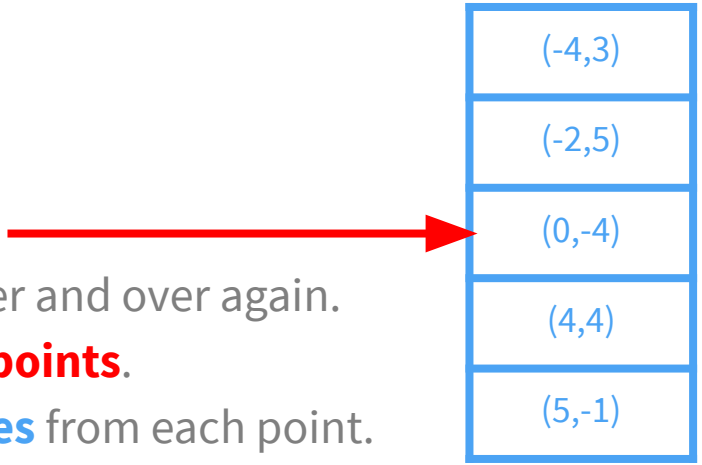
1. Click on the Green Flag to run the program.



Walk through the Graphing Coordinates

The instructions in the program...

1. Asks you for your name
2. Tells you to **click these 5 points** on the graph
    a. The program **only asks to click (-4, 3)** over and over again.
    b. The program **doesn't actually mark the points**.
3. Next the program ***should* ask you for the slopes** from each point.
    a. The program **doesn't progress on to the slopes activity**.

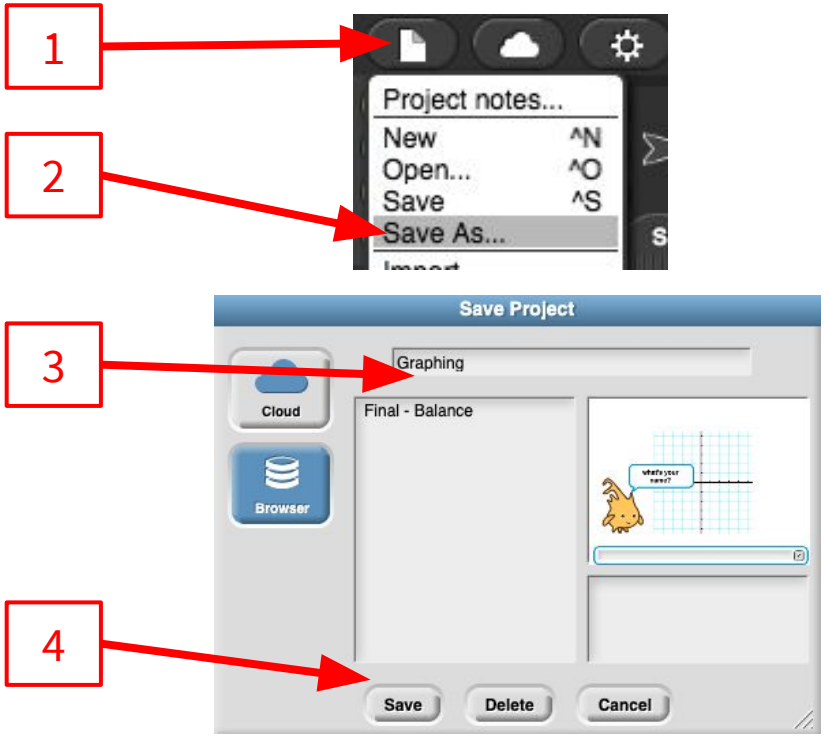


Let's work on how to make this **starter code** work.

Save Your Code Before Start

It's a good practice to Save your code every time you finish something!

1. Click the **File Icon**
2. Select "**Save As...**"
3. Type in "**Graphing**"
4. Click on the **Save** button



Step 1: Progress to the Slope Activity

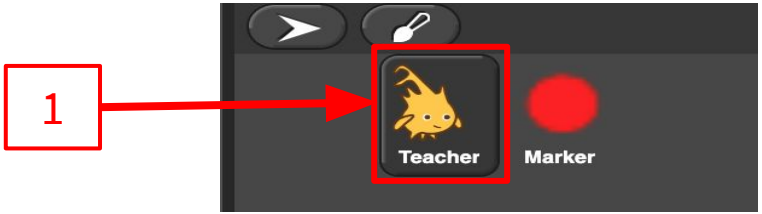
First, let's see how to progress to the Slope activity.

1. Click on the Teacher sprite
2. Find the slope activity script.

It seems that the code that handles the slope activity is already provided.

So we just need to use a 

block to trigger the slope activity event.



Step 1: Progress to the Slope Activity

Let's add the broadcast block.

1. Follow the comment in this code to add a broadcast block for "activity: slope"

Incomplete Code

Check your answer

Step 1: Progress to the Slope Activity

Is this what your code looks like?

Now click on the Green Flag button to play through the activity again. See if the slope activity appears.

Abstractions

Notice the **Introduction** block is a custom block that **abstract** away the details of the introduction code.

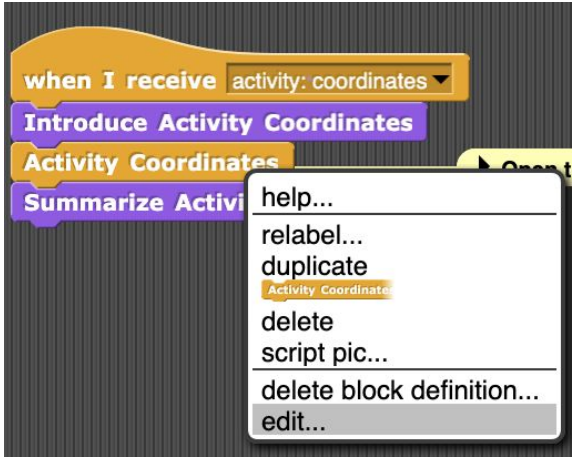
Save your code!

Step 2: Ask all Points

Remember how the Teacher Sprite would ask to click on point (-4,3) over and over again even when you have clicked on the right point?

There is a bug!!!

1. Stay on the **Teacher** Sprite
2. Right click and edit the **Activity Coordinate** block



Step 2: Ask all Points

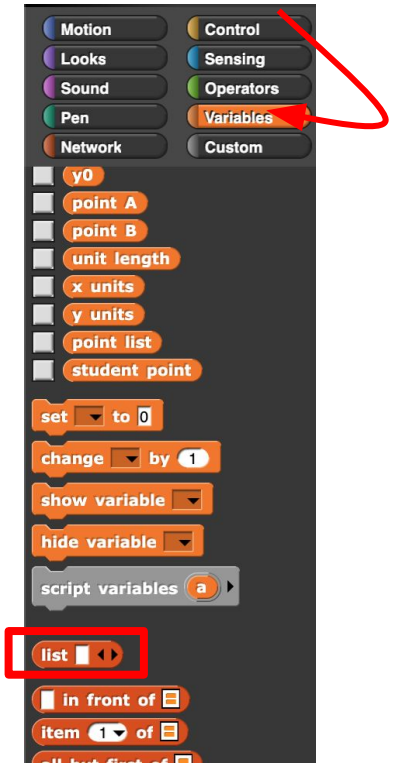
Before we start, we need to know how this project stores the x and y coordinate of points!

A **coordinate point** (x,y) is stored as a **list**. You can find list blocks in the **variable** category of the palette.
For example, here is coordinate point (-4,3): 

Point A is coordinate point (-4,3): 

The x coordinate of point A is: -4 

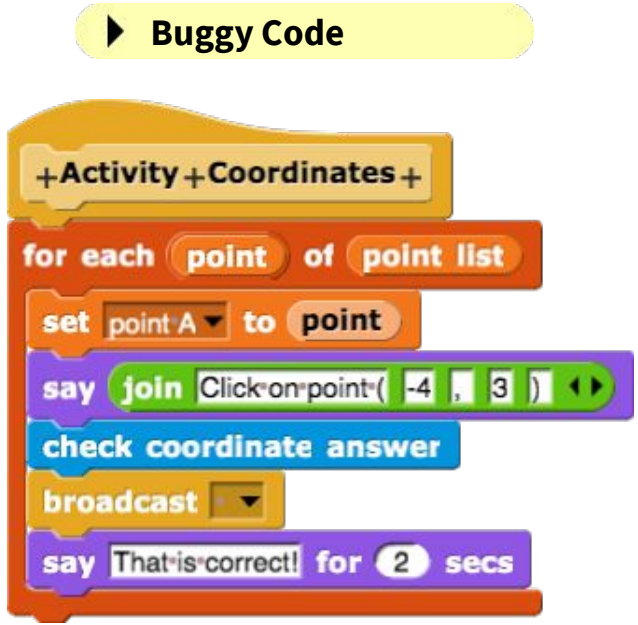
The y coordinate of point A is: 3 



Step 2: Ask all Points

Notice how, **for each point in our point list**, the script always **says** point (-4, 3) no matter what the **point** is?

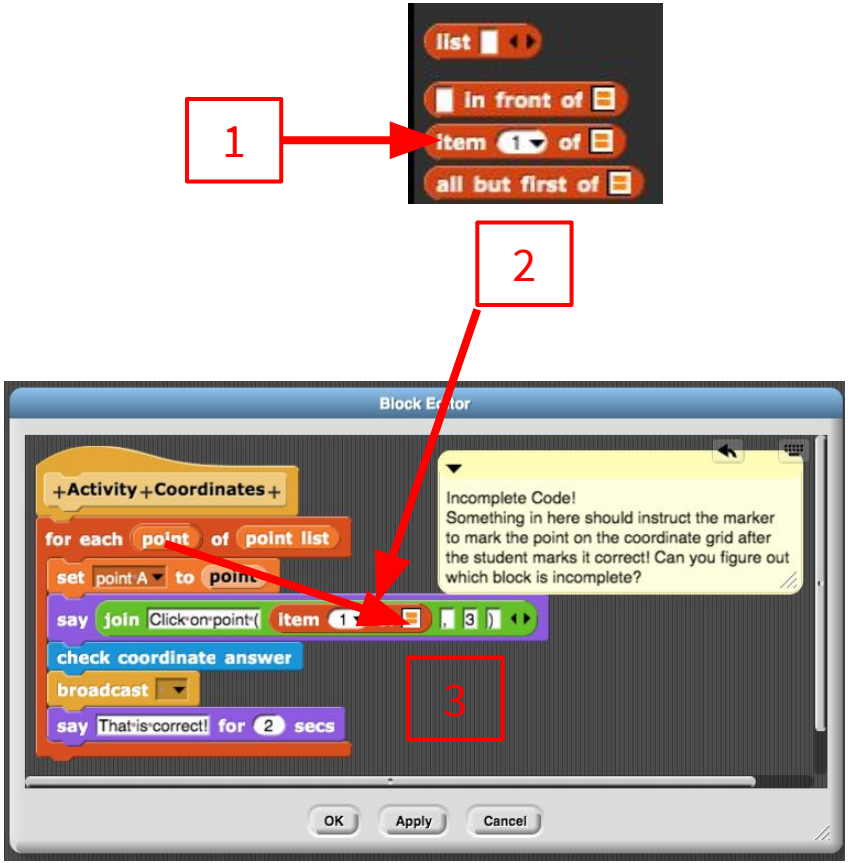
This is smells buggy!



Step 2: Ask all Points

Let's make the say block dynamic!

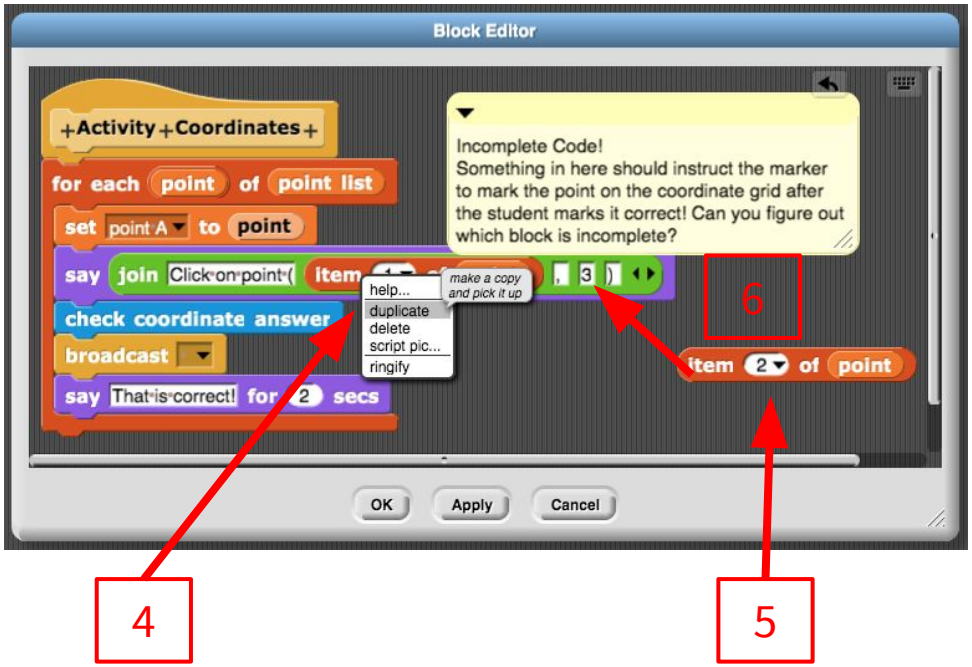
1. Find the item 1 of block from the palette under Variables tab
2. Drag the item 1 of block and drop it in the place of "-4"
3. Drag the **point** block and drop it in the second slot of the item 1 of block. (represents **list** in Snap)



Step 2: Ask all Points

4. Right click on the 

block, choose "duplicate", and place the duplicated block aside
5. Make sure you change the item number to "**2**".
6. Drop the duplicated block into the number "3" slot.
7. Click on the Apply button (don't close the block editor yet)



Check your answer

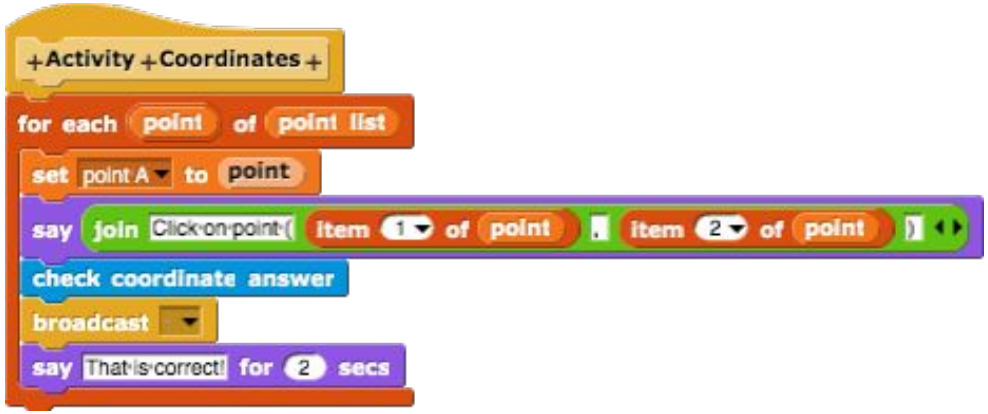
Step 2: Ask all Points

Did you figure it out?

Awesome!

Now that you **fixed the bug** in the code, play through the activity again and see if the program **asks different points**!

You're doing great!



Save your code!

Switch for the next activity!

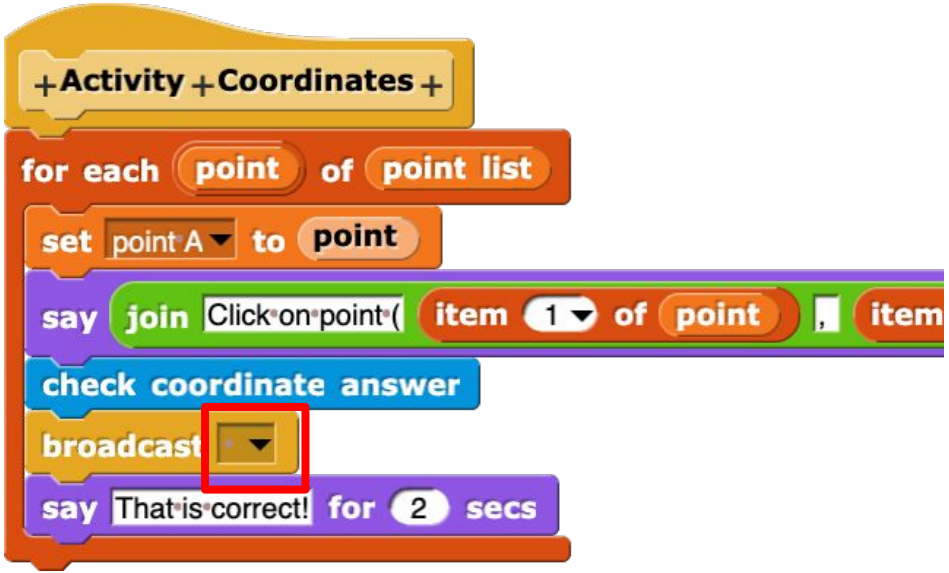
Teacher A: Drive!
Teacher B: Navigate!

Step 3: Mark the points

Now the Teacher sprite is asking to click on different points, but it still doesn't mark it when I get it correct!

It is because the code is **incomplete**!

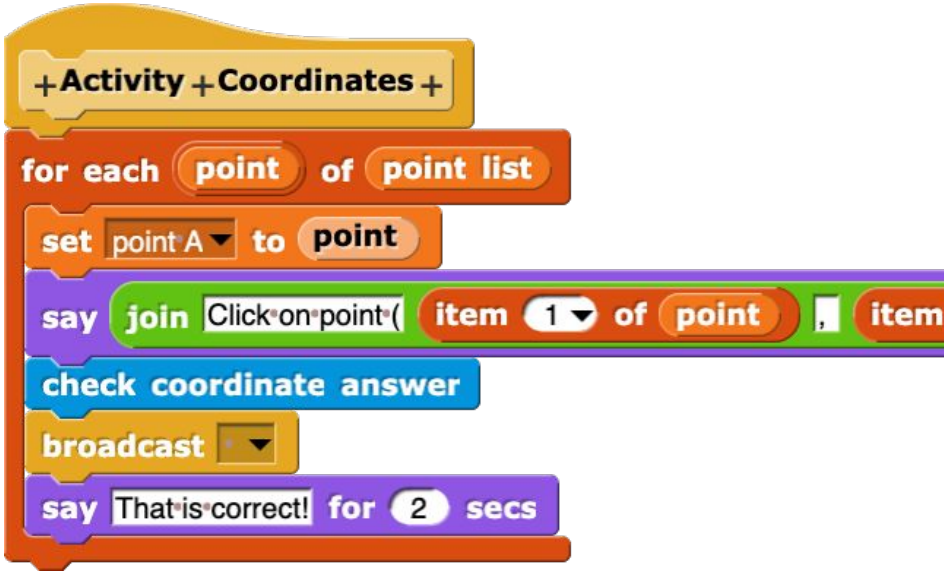
Notice how the **broadcast** block is empty?



Step 3: Mark the points

1. Program the script to broadcast a message so we can mark the point.

Hint: You shouldn't need to add a block - only modify a block that is already in the script.

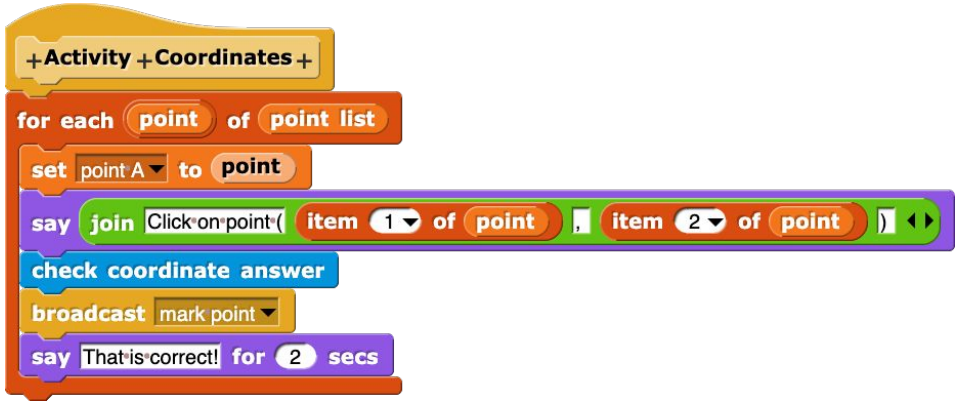


Check your answer

Step 3: Mark the points

Is this what your code looks like?

Now that you **completed** the code, play through the activity again and see if the program **marks the points**!



Pattern Recognition

Notice how this custom block uses a for each loop to **repetitively** ask about each point!

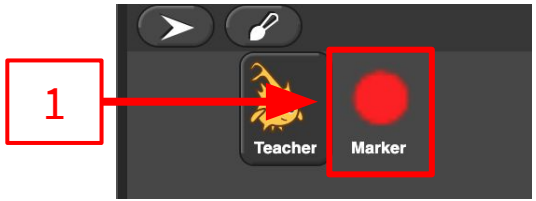
Algorithms

You've just finish an **Algorithm** that asks user to click on the points in the coordinate grid!

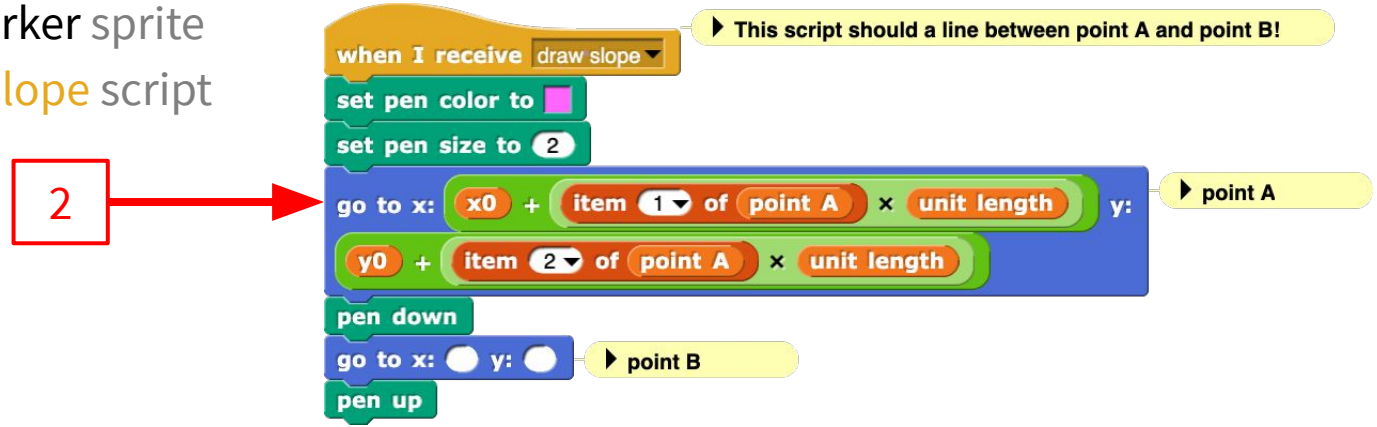
Save your code!

Step 4: Draw the Slopes Correctly

If you have played your code recently, when it draws the slope, it draws them incorrectly. Let's work on that!



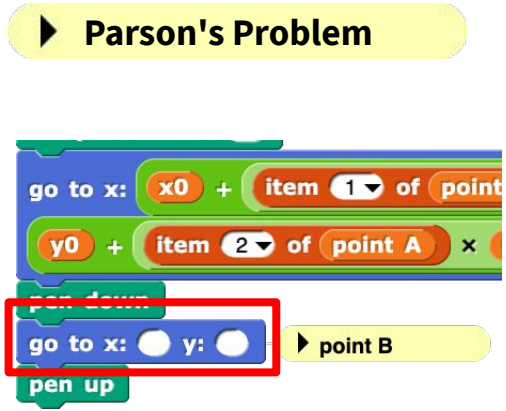
1. Click on the Marker sprite
2. Find the draw slope script

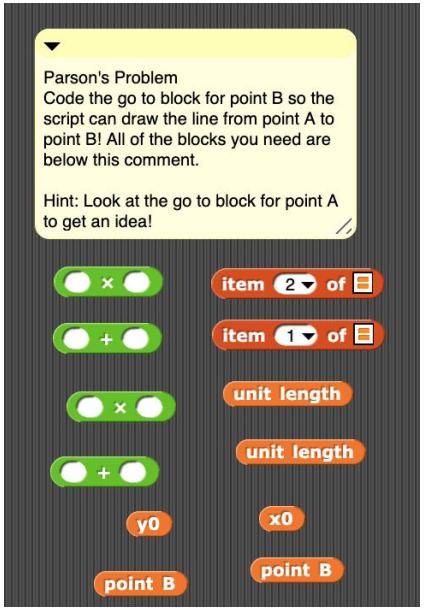


Step 4: Draw the Slopes Correctly

This script should draw a line from point A to point B !

1. Use the available blocks on the right to **complete** the **go to x: _ y: _** block for point B





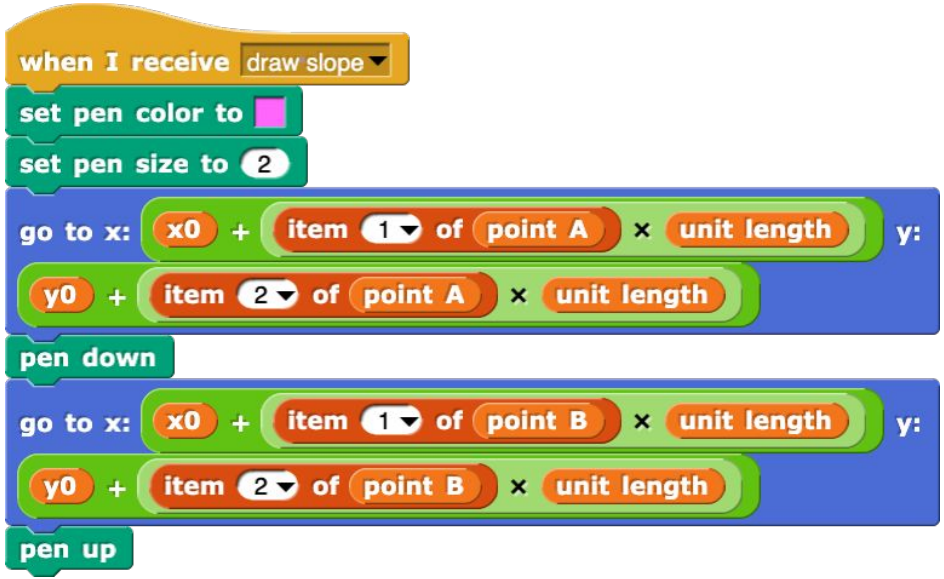
Check your answer

Step 4: Draw the Slopes Correctly

How does your code compare?

Now that you **completed** the code, play through the activity again and see if the program **draws the slopes correctly**!

Awesome!



Save your code!

Switch for the next activity!

Teacher A: Navigate!
Teacher B: Drive!

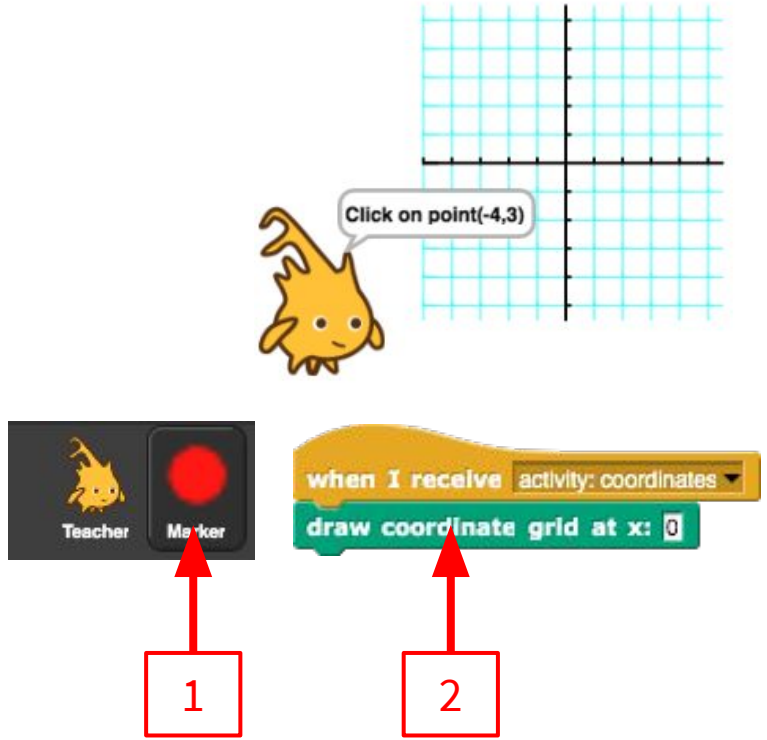
Step 5: Move the Coordinate Grid Right

When you run the code, do you find the Teacher's words covers the grid making it hard to click on the point?

Let's see how we can move the grid to the upper right corner.

1. Click on the **Marker** sprite
2. Find this script

(If you need reminder of the position in the Stage, click here)

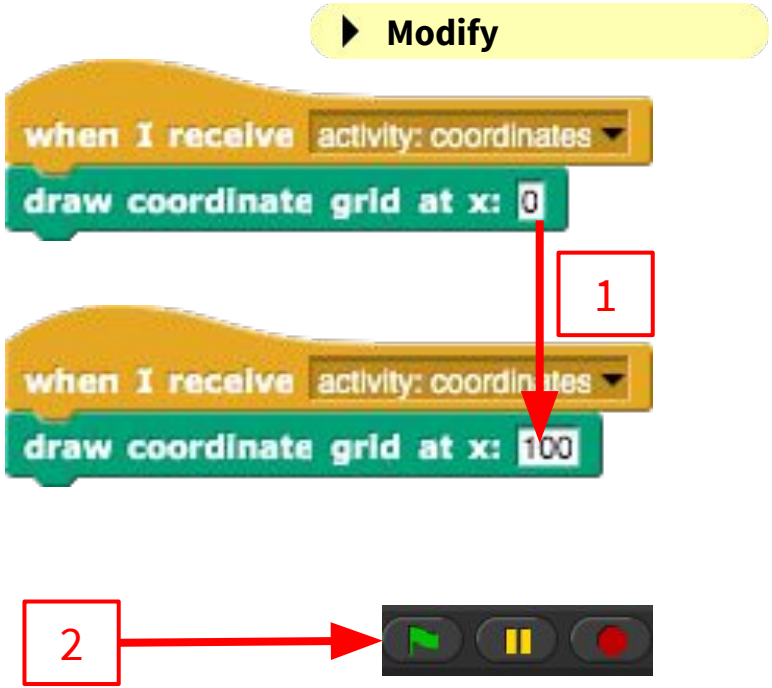


Step 5: Move the Coordinate Grid Right

It seems that we can change the **x position** of the grid.

1. Change the **x position** from 0 to 100.
2. Click on the **Green Flag** to run the program again.

Hey, the coordinate grid **moved to the right**!



Save your code!

Step 5: Move the Coordinate Grid Up

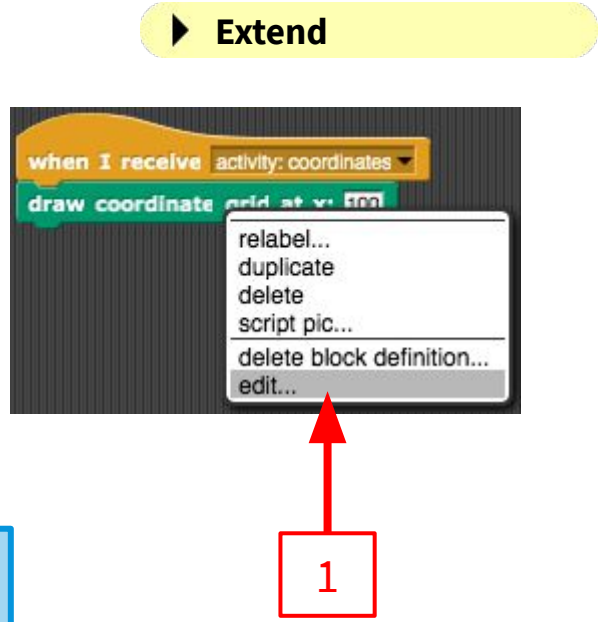
We can move the coordinate grid left and right.
How do we move it up and down?

It seems like we need to add a slot to the **draw coordinate grid** block to specify the y position!

1. Right click and edit the **draw coordinate grid** block

Abstractions

Let's look into what is **abstracted** away in the **draw coordinate grid** block.



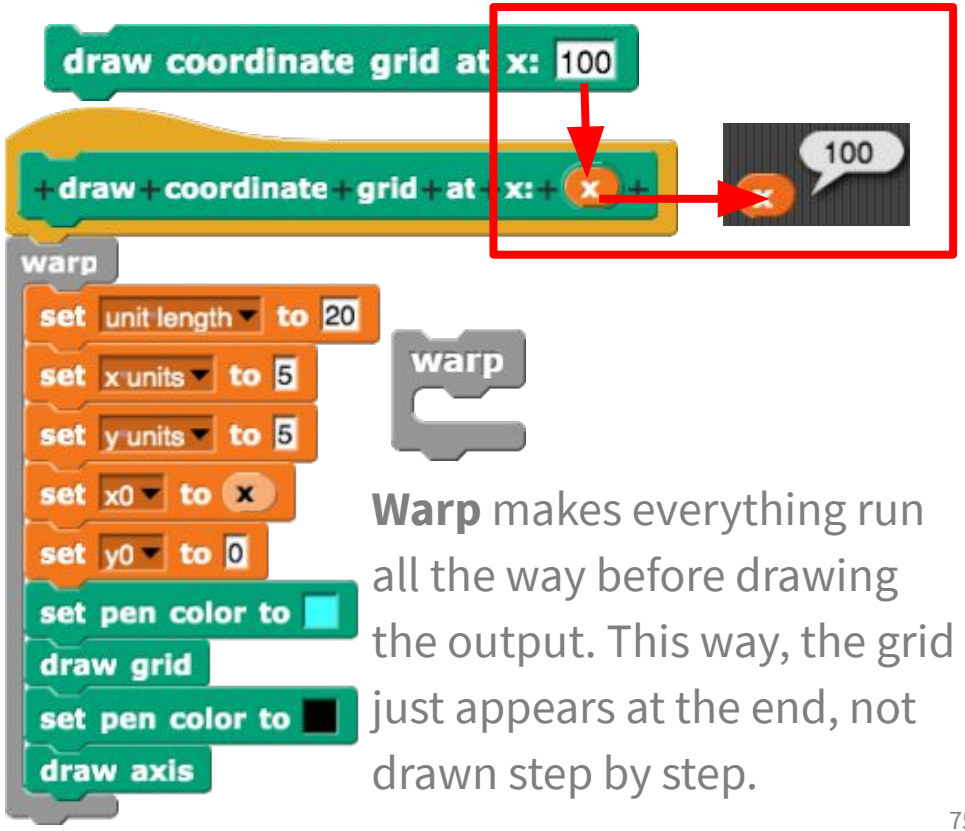
Step 5: Move the Coordinate Grid Up

Notice the (x) block in the name of the **draw coordinate grid** block?

It is called a **parameter** block. The parameter block creates a **textbox** in the **block name**.

The **parameter** block **holds** whatever **value** the user put in the **blank** (e.g. number, word, answer).

Parameter block allows more **flexibility** and **reusability** of a custom block.



Warp makes everything run all the way before drawing the output. This way, the grid just appears at the end, not drawn step by step.

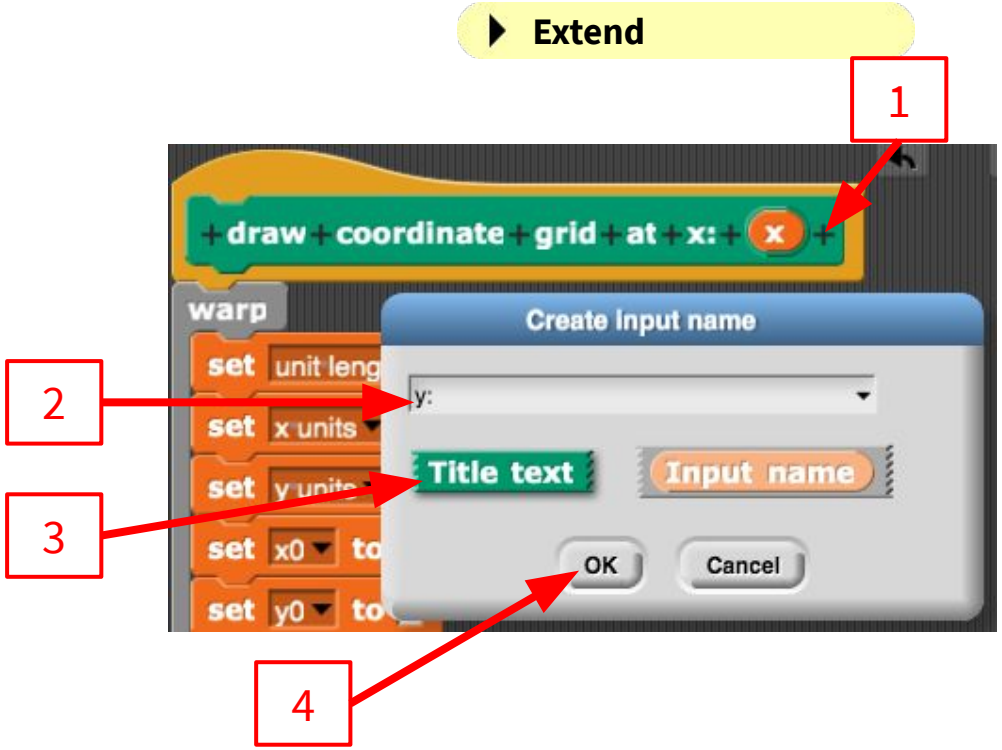
Step 5: Move the Coordinate Grid Up

Let's add "y: (y) " to the block name.

Let's add the "y:" part first:

1. Click on the **right most** "+".
2. Type "y:" in the text box.
3. Click on the "**Title text**".
4. Click OK button.

Now "y:" should show up in the block name.

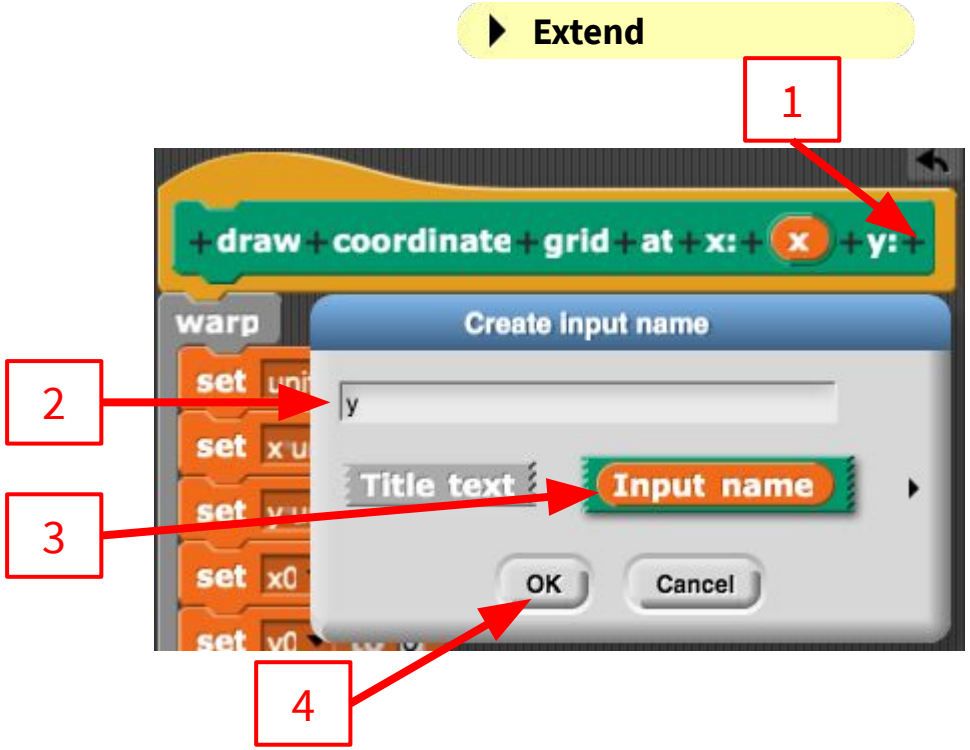


Step 5: Move the Coordinate Grid Up

Let's add the (y) parameter block:

1. Click on the **right most** "+".
2. Type "y" in the text box.
3. Click on the "**Input name**".
4. Click OK button.

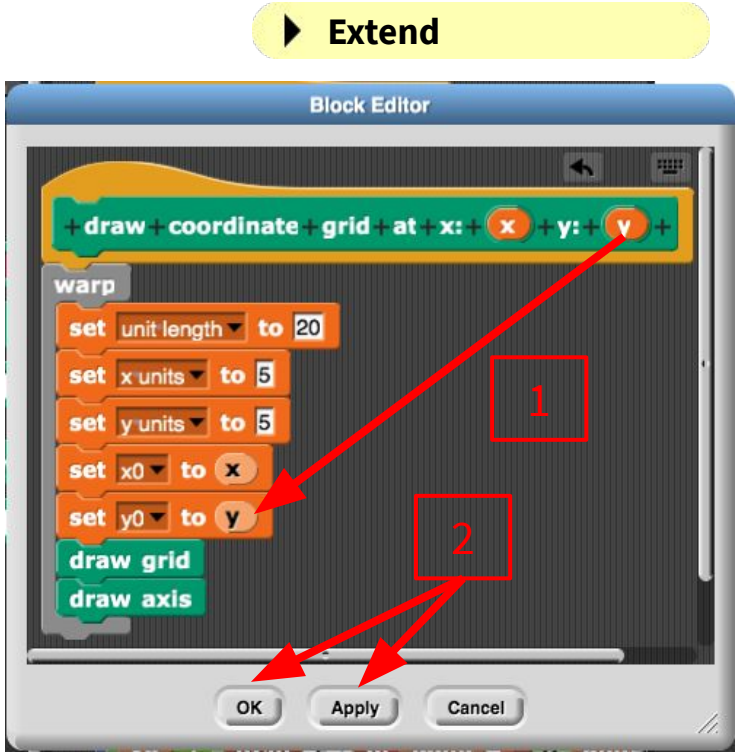
Now (y) should show up in the block name.



Step 5: Move the Coordinate Grid Up

Now that we've added the y parameter block, we need to use it in the draw coordinate grid block.

1. Drag and drop the **y** parameter block in the title into the "**set y0 to _**" block.
2. Click on the Apply button and **then** OK button.

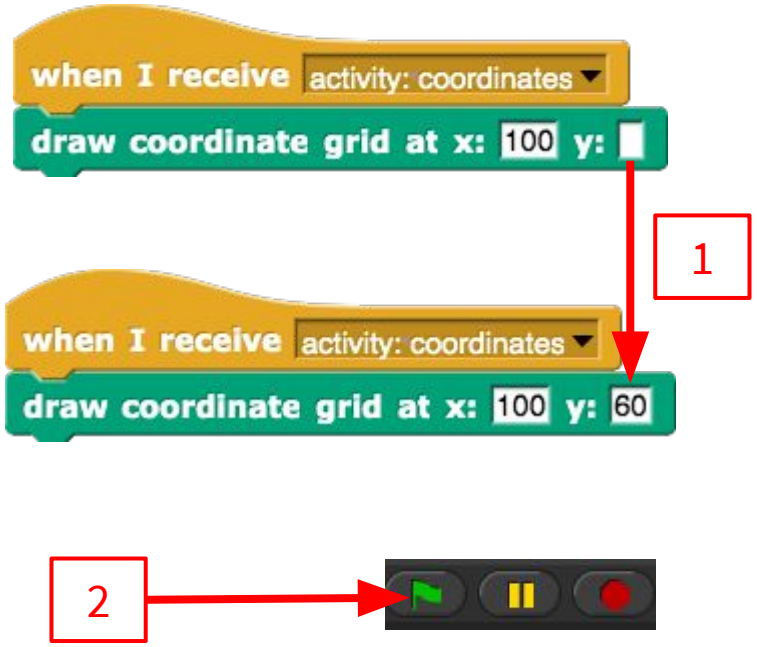


Step 5: Move the Coordinate Grid Up

Now, you should see **a slot for y position** in the **draw coordinate grid** block.

1. Change the y: slot from empty to 60.
2. **Click on the Green Flag** to run the program again.

Now the coordinate grid is in the **upper right corner**. Yay!



Save your code!

If you have time...

Do you see a **Computational Thinking** PRADA element in this sequence?

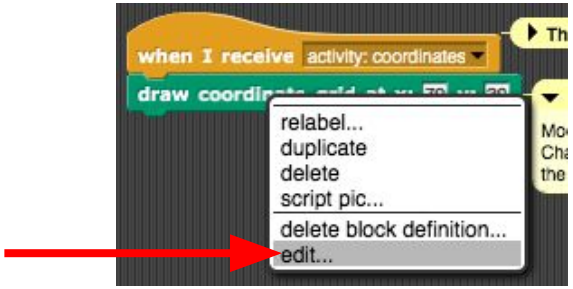
Switch for the next activity!

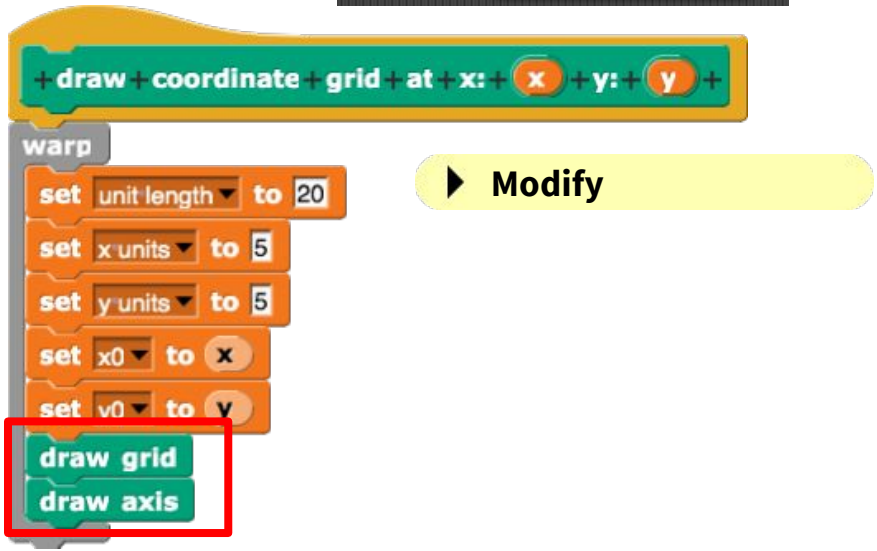
Teacher A: Drive!
Teacher B: Navigate!

Step 6: Change the Coordinate Grid Size

What else can we change to make the coordinate grid look different?

1. Right click and edit the **draw coordinate grid** block
2. Examine the script in the block.





Abstractions

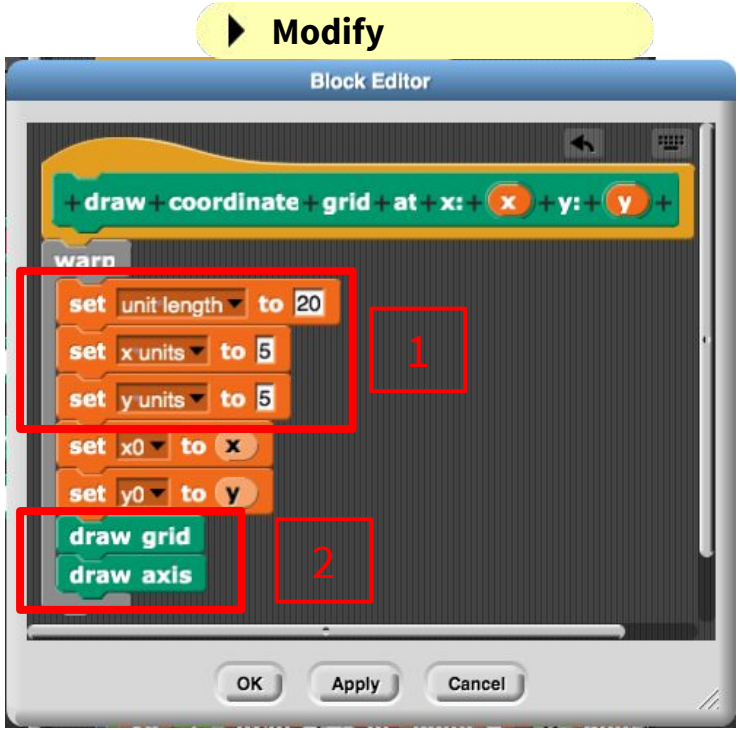
Notice how the **draw coordinate grid** block has two custom blocks **draw axis** and **draw grid** that **abstracts** away more details?

Step 6: Change the Coordinate Grid Size

Let's explore how we can change the look of the coordinate grid.

1. Change the numbers in the set blocks and see how the size of the coordinate grid changes.
2. Look inside draw grid and draw axis grid to see how to change the **color** and **thickness** of the coordinate grid.

Don't forget to hit **Apply** after changes.

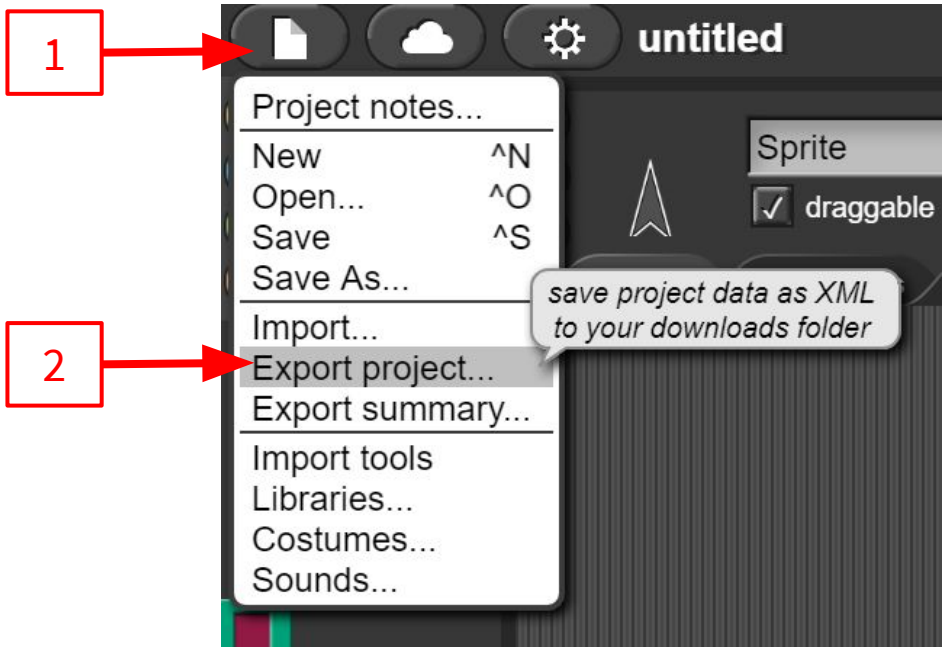


Save your code!

You've Completed Activity 2!

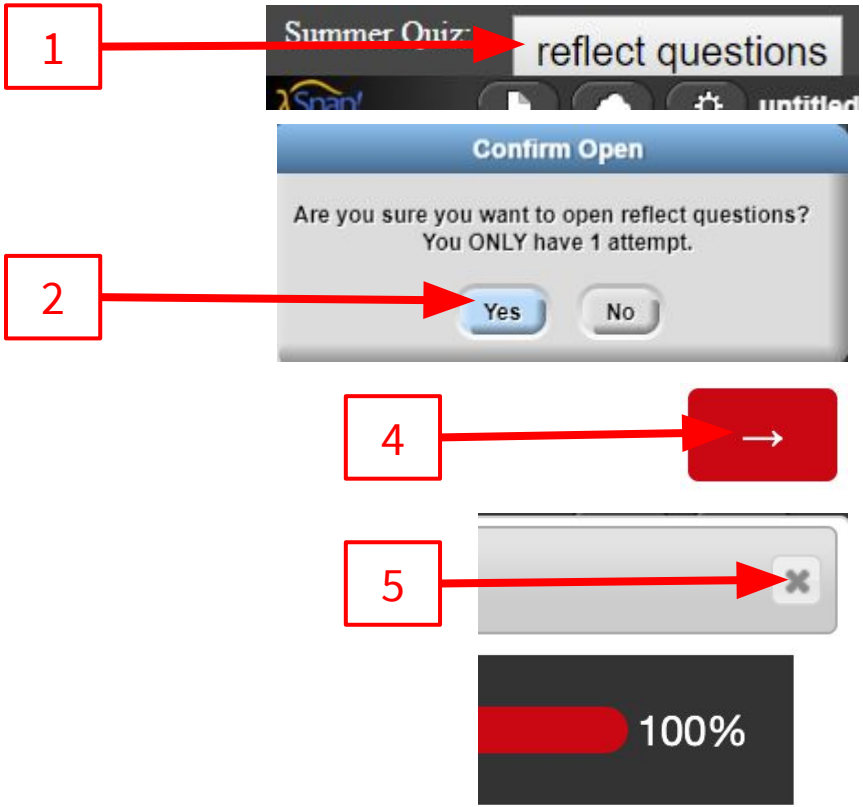
If you want to export and download the project to your computer, here is how:

1. Click on the File Menu
2. Select on Export project
3. Choose where you want to save the project file and click save



Let's Reflect!

1. Click "Reflect Questions" in the **top right**.
2. Press **Yes** to begin the reflection.
3. Use the **red arrow button** to go to the next question.
4. Finish out the reflection quiz.
5. After finishing the quiz, use the **cross button** on the top right corner of the survey window to close.

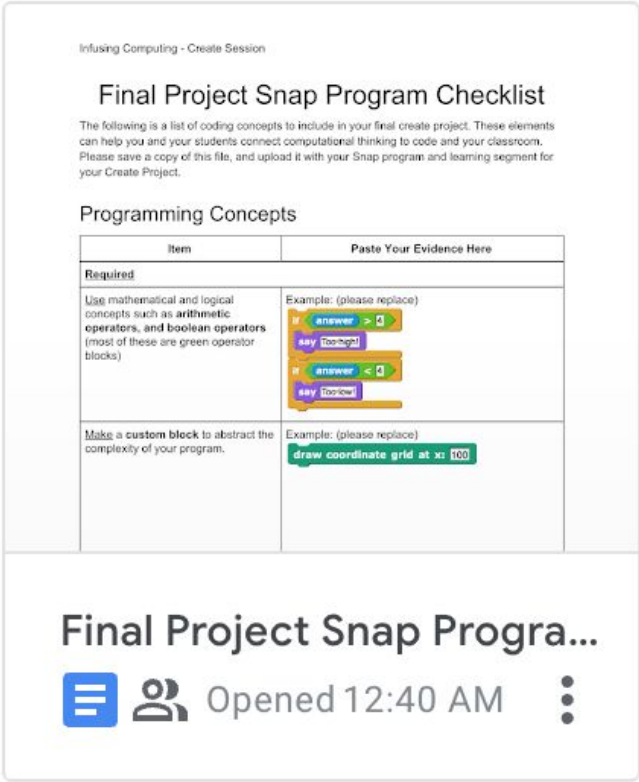


Final Snap Program Checklist

Final Snap Program Checklist

At the end of the PD, we would like each team to submit *at least one* **Final Snap Program Checklist** along with your **Lesson Plan** and **Snap Project File**.

The Final Snap Program Checklist is **a tool** to help you make sure that you have some of the **important programming concepts** and **PRADA elements** in the Final Snap Project File(s) that you submit at the end.



Final Snap Program Checklist

This **checklist includes three sections**:

- ➢ Programming Concepts
- ➢ PRADA Elements - CT
- ➢ Project Design

Each section will have a set of requirements.

You will find that a lot of the requirements **have appeared** in either the **coding activities** or in the **reflect questions** multiple times!

Programming Concepts

Item	Paste Your Evidence Here
Required	
Use mathematical and logical concepts such as **arithmetic operators, and boolean operators** (most of these are green operator blocks)	Example: (please replace) 

PRADA Elements - Computational Thinking

Item	Paste Your Evidence Here
Required	
Add a comment to a code segment that can teach **Pattern Recognition** and use your own words to explain how it represents pattern recognition *in your context*.	Example: (Please replace) 

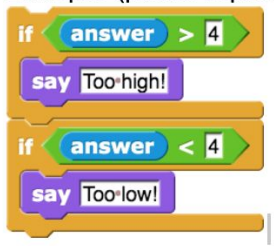
Project Design

Item	Paste Your Evidence Here
Required	
Write a short paragraph to describe what your code project does or what would you like your code to do if not all the features are finished.	Example: (Please replace) This project is an interactive narrative that introduces me and my family. It includes three parts. In the first part, the project will greet the player by name. Then it talks about myself and each of my family members. Next, it plays a guessing game with the player where the player is asked to guess how many years I've been

Checklist - Programming Concepts

In the Programming Concepts section, you'll need to **paste a picture of the script** that has the required programming concepts. An example is shown below. You can find how to **get a clear script picture** by looking at the FAQ (click here).

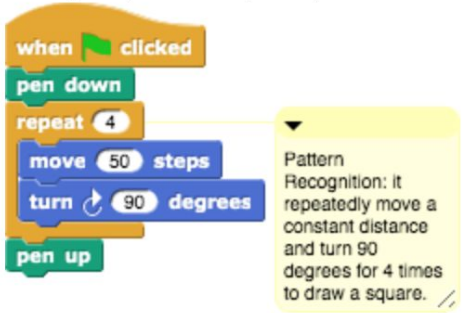
Programming Concepts

Item	Paste Your Evidence Here
Required	
Use mathematical and logical concepts such as **arithmetic operators, and boolean operators** (most of these are green operator blocks)	Example: (please replace) 
Make a **custom block** to abstract the	Example: (please replace)

Checklist - PRADA Elements

In the PRADA elements section, you'll need to **paste a picture of the script along with descriptive comments** of each of the four PRADA elements. An example is shown below. You will get practice on how to **insert comments** in Day 3.

PRADA Elements - Computational Thinking

Item	Paste Your Evidence Here
Required	
Add a comment to a code segment that can teach **Pattern Recognition** and use your own words to explain how it represents pattern recognition *in your context*.	Example: (Please replace) 

Checklist - Project Design

In the Project Design section, you'll need to **write short paragraphs** to describe what your code does and what kind of coding activities did you plan for your students! An example is shown below.

Project Design

Item	Paste Your Evidence Here
Required	
Write a short paragraph to describe what your code project does or what would you like your code to do if not all the features are finished.	Example: (Please replace) This project is an interactive narrative that introduces me and my family. It includes three parts. In the first part, the project will greet the player by name. Then it talks about myself and each of my family members. Next, it plays a guessing game with the player where the player is asked to guess how many years I've been teaching and it will provide feedback on if the guess was too high or too low until the player gets it correct. Finally.....

Final Snap Program Checklist

Please don't panic!

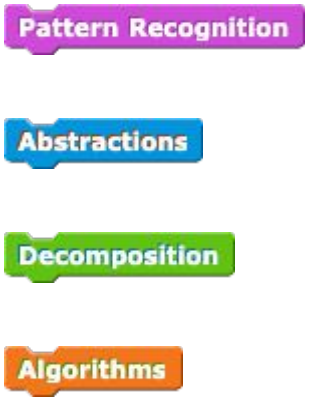
We'll make sure you **get enough practice** in the code session to **help you fill out** everything in the checklist.

You can find a **template** of the Snap Program Checklist by clicking here.

It would be very helpful to keep these requirements in mind when you're designing your own CT infused project for your lessons.

Reflection

We saw PRADA Concepts in action



We also learned Snap*!* Concepts

How to duplicate sprites and scripts

How to use lists and check sprite interactions

How to use parameters in custom blocks

Different activity types - (Extensions, incomplete code, pseudo code)

Congratulations on your coding conquest!

3C