Welcome to the ODU ALPR Microcredential Course!

IMPORTANT! Please read this message before you begin the coursework for this microcredential.

Welcome to the microcredential course on Algorithms and Programming! We are using the abbreviation ALPR to differentiate this course from the others, so you will see that acronym used throughout the course.

Please note that though this course is formatted similarly to the other content courses in this microcredential stack, the content will be presented differently. We have provided tutorials and activities along with videos to support comprehension of the content for this course. We recommend that you download the slides and video transcripts prior to viewing the videos to support comprehension.

You will notice that there are more learning modules in this course than in the other content courses in this stack. The modules in this course will focus on a theme or concept, some of which may be new or more challenging. Therefore, we are presenting the content in smaller, more digestible pieces.

Also different in this course is that the quizzes for each module will serve as the formal assessment of the content for those sections, so there will be no final assessment in this course.

This microcredential course is formatted into two sections:

- Course Materials, comprised of Learning Modules containing content and resources for competencies that are aligned with the Virginia Computer Science SOLs and assessments
- Lesson Plan Assignment, including resources and templates for developing your lesson plan for the course

To maximize development of your computer science pedagogical content knowledge, we recommend that you step through each module in the order presented. **Access the Course Materials and Lesson Plan Assignment using the menu to the left.**

Once you have completed each of the learning modules for this course, you will complete the Lesson Plan Assignment.

Here is what to do:

- Click sequentially through the modules in Course Materials, view the materials and complete the activities where applicable.
- Complete the assessment at the end of each module to show evidence of mastery of that module's content.
- Upload a lesson plan to finish the microcredential.

If you need assistance at any time, please email TCEP@odu.edu.

Algorithms and Programming Modules

Algorithms and Programming Module 1 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-1)

This learning module will focus on **Scratch**.

This module is not aligned with specific teacher competencies but is designed to provide an introduction to the coding language **Scratch,** which will be used in subsequent learning modules in this course.

Algorithms and Programming Module 2 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-2)

This learning module will focus on **Sequences**.

When you complete this learning module, you should be able to:

- **Construct** a sequence of steps (algorithm) to accomplish a task.
- **Identify** the beginning, middle, and end of a sequence.

Algorithms and Programming Module 3 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-3)

This learning module will focus on **Algorithms**.

When you complete this learning module, you should be able to:

- **Construct** algorithms to include structures such as loops, variables, and conditionals.
- **Explain** different types of creative products that can be generated using a computing device (e.g., computer games, interactive stores, graphic design, programs, music, and movies).
- **Determine** an original problem and create a solution using a text or block-based program.

Algorithms and Programming Module 4 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-4)

This learning module will focus on **Classification**.

When you complete this learning module, you should be able to:

- **Classify** and **arrange** items based on their attributes, actions, and patterns.

Algorithms and Programming Module 5 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-5)

This learning module will focus on **Debugging and Fixing**.

When you complete this learning module, you should be able to:

- **Describe** how an algorithm didn't work (e.g., character is not moving as intended)

- **Analyze** a flawed algorithm to determine possible solutions.
- **Implement** a proposed adjustment to an algorithm that wasn't working as intended.

Algorithms and Programming Module 6 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-6)

This learning module will prepare you to **Plan and Decompose**.

When you complete this learning module, you should be able to:

- **Design** a program using a planning tool.
- **Describe** how an iterative design process can improve an algorithm.
- **Analyze** and **decompose** a problem into subproblems.
- **Explain** why multiple smaller problems may be easier to solve than one large problem.

Algorithms and Programming Module 7 (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-module-7)

This learning module will focus on **Giving Credit to Sources**.

When you complete this learning module, you should be able to:

- **Review** a program written by a programmer and identify portions that may have been created by others.
- **Explain** why it is important to give credit to authors.
- **Describe** when it is acceptable to use people's work, and how to give credit to sources.

Algorithms and Programming Module 1

This learning module will focus on **Scratch**.

This module is not aligned with specific teacher competencies but is designed to provide an introduction to the coding language **Scratch,** which will be used in subsequent learning modules in this course.

Let's Learn Scratch

View the videos in the order presented on each of the following pages. Walk through each of the three lesson sections sequentially.

A PDF of the slides and a transcript of the videos are also included for accessibility.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

Let's Learn Scratch - Part 1

INTRODUCTION TO SCRATCH

Slide 1
Hello, and welcome to the ARCS Microcredential of Algorithms and Programming. This is an introduction to Scratch.

What is Scratch?

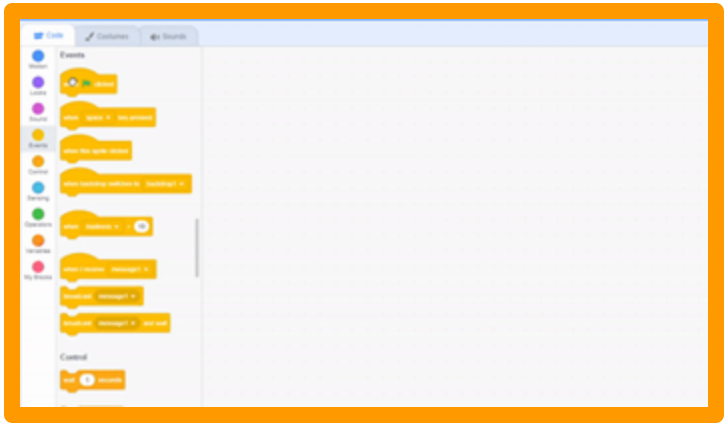
Scratch is a **block based** programming language.

This means, **instead of** `> typing` ,

you ***drag*** blocks in order to build code!

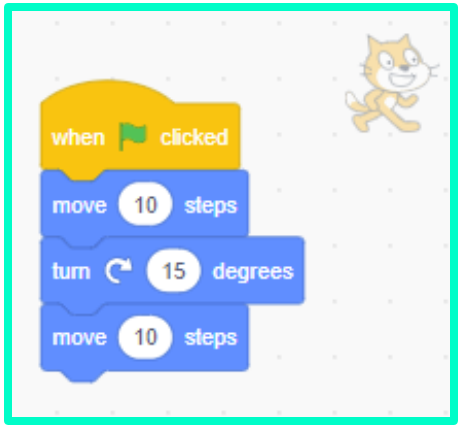
- Visit the environment **here!**

Tip: When you visit the Scratch environment, there will be a cat sprite by default.

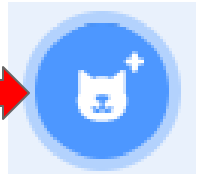


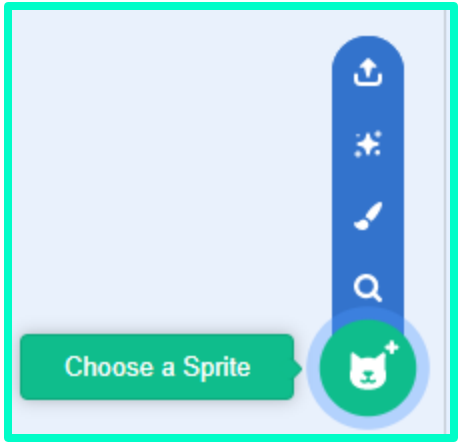
Slide 2
What is Scratch? Scratch is a block-based programming language. This means, instead of typing, you drag blocks in order to build code. As you can see here, in this small bottom right corner, you go to a code block and drag it into a coding area. With this PowerPoint, I've gone ahead and added a hyperlink to the environment where you will be programming. Here's a tip: when you visit the Scratch environment, there will be, by default, a cat Sprite.

What is a Sprite?



- SPRITES are little *characters* that **store** your code!
- There are many `preloaded` **options** for sprites.
- **Add sprites** to your project using the Choose a Sprite button.
  - This brings up the **menu** of sprites
  - From there you can **select a sprite** to add to your project

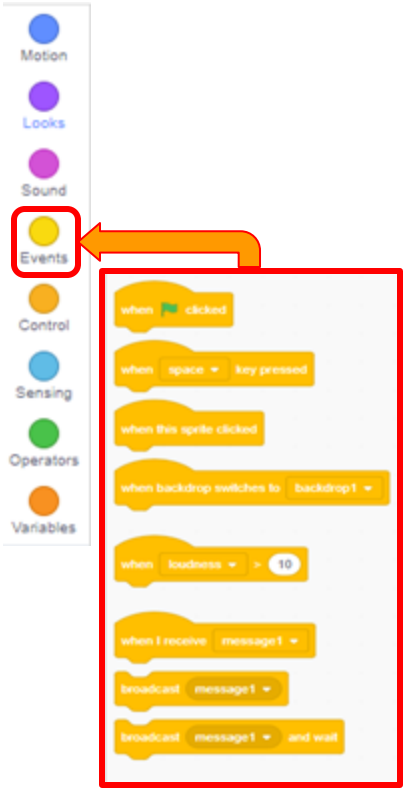




Slide 3

Speaking of which, what is a Sprite? Sprites are little characters that store your code. There are many pre-loaded options for Sprites. You can add a Sprite to your project using the "Choose a Sprite" button. It looks like this circle with a cat head and the little plus sign. This brings up the menu of Sprites, and from there you can Select the Sprite to add to your project. This is what a script or code block section would look like as being "stored" in your Sprite.

Programming your Sprite – EVENTS !



All code blocks are grouped by their effect!

A good place to start is **event blocks**.

They **declare the event** that must occur for your code to *RUN*.

DRAG and **DROP** the block of your choice **!**

The **easiest** to start with is the "**when Green Flag clicked**" block.

*It's possible to have **more than one** event block !*

(WE'LL STICK WITH ONE FOR NOW)

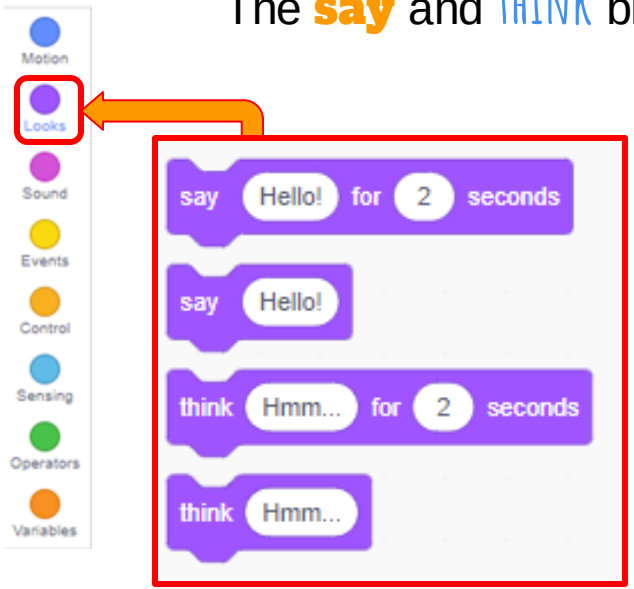
Slide 4

Let's talk about how to program our Sprites. We're going to talk about events, specifically. In Scratch, all code blocks are grouped by their effect. A good place to start is event blocks. They declare the event that must occur for your code to run. If you look on the right, you can see the different groupings. There's motion blocks, looks blocks, sound blocks, event blocks (which we are now discussing), control blocks, sensing blocks, operators and variables.

Programming your Sprite – LOOKS!

Looks blocks allow for visual changes to sprites and backdrops.

The **say** and THINK blocks allows for **text** to be displayed above the sprite.



say blocks = ***speech*** bubble

THINK blocks = ***thought*** bubble

Slide 5
And next we're going to talk about looks. Looks block allow for visual changes to sprites and backdrops. The say and think blocks allow for text to be displayed above the Sprite. Say blocks are equivalent to a speech bubble and the think blocks is equivalent to a thought bubbles. The looks are in this look section are "Say Hello for [two] seconds" and "Say Hello".

The difference between the 'say Hello for two seconds' and the 'say Hello' block are that the say Hello for two seconds block displays the word 'Hello', pauses the program for two seconds, and then stops displaying 'Hello' as the program continues. The simple 'say Hello' block will display our speech bubble with the word 'Hello', but will not stop displaying until specified later in the program. And the same is for 'the think for two second' and simply 'think'. In the next video, we're going to get started with our project: Intro 1.

Project Intro 1 - Hello World

1. **Select a sprite** from the sprite selection menu
2. **Drag** the "**when {green flag} clicked**" event block into the code area
3. **Choose a think or say block** and attach it to the **bottom** of the event block
4. Enter "**Hello World**" in the text box
5. **Click** the green flag
6. **Save** your project

Slide 6
Hello, everyone. Welcome to the Project Intro 1 - Hello World. Our first step is to select a Sprite from the Sprite selection menu. Next, we're going to drag the "when green flag clicked" event block into the code area. Third, we're going to choose a think or say block and attach it to the bottom of the event block. Then we are going to enter 'Hello World' in the text box. Lastly, we're going to click the green flag and save our project. This is the point of video where you can pause and attempt to do this by yourself, or wait and we can do it together, or you simply just watch me do it. Let's get into this.

We're now in the Scratch coding environment. Our first step was to Select the Sprite. Instead of using the default cat, I'm going to go ahead and delete the cat by clicking on this little trash bin here. I'll go into our button that we introduced earlier to choose the Sprite and click on it. As you can see, there's a large selection of Sprites. I'm going to go with the crab as my Sprite. Our second step is drag the "when green flag clicked" event block into the code area. The keyword there is 'event' block. We're going to come over here and see where it says events, as it's going to automatically take us to where the event blocks are.

I'm going to grab the "when green flag clicked" block, then drag and drop it into the code area. Our third step is to choose a think or say block and attach it to the bottom of the event block. The think or say blocks exist in the looks section, so we're going to click the purple circle called 'looks'. Then we can see our 'Say Hello' or 'Think'. I'm going to choose the 'Say Hello for two seconds' block, and then we're going to enter 'Hello World' in the text box. Where it says 'Hello', we can click on the field and type 'Hello World'. Here; let me blow that up a little bit. Our last step is to click the green flag. Finally, we're going to save our project. In order to do so, we have to create an account. Most of you probably do not have a Scratch account. Simply, come up to the top right corner, click 'Join Scratch', create an account, and then you can click Save. Mine's says 'project saved' when I automatically log in. That concludes Project Intro 1 – Hello World.

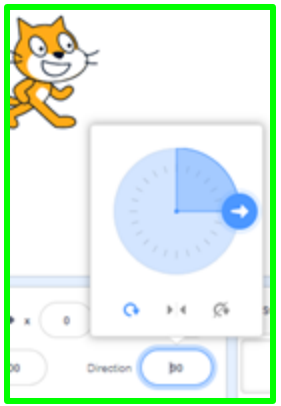
Now that we've completed our first project, let's go ahead and learn some more.

Programming your Sprite – MOTION!

Motion blocks create motion in your program.

They **move your sprite** in the direction it is pointing.

The sprites exist on a **grid**, with each "**step**" changing it's coordinates!



Turn blocks change the sprite's *direction*.

- It rotates **clockwise** or **counter-clockwise**

Go to blocks move the sprite *instantly* to a chosen grid position

- It can "go to" specific **coordinates**, other **sprites**, or a **random position**

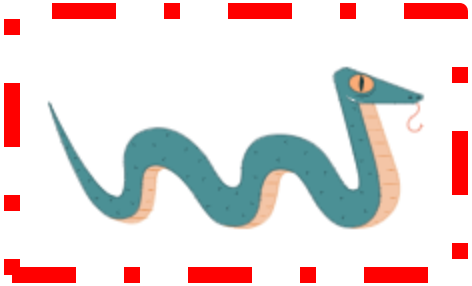
Glide blocks are similar to go to blocks, but the move is *not instant*

- The sprite "**glides**" **across the grid** instead of instantly repositioning

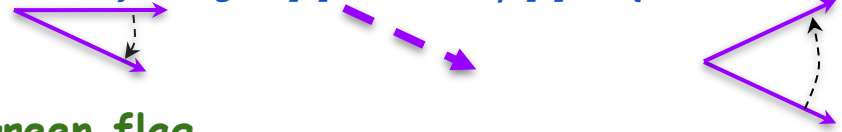
Slide 7
With these Sprites, we're going to learn about motion. Motion blocks create motion in your program. How? They move your Stripe in the direction it's pointing. Specifically, the move block is the one that does this. The sprites exist on a grid, with each step changing its coordinates. 10 steps would move it 10 "coordinates" depending on which direction it's pointing. The turn blocks change the direction the Sprite is facing. It rotates clockwise or counterclockwise. Here you can see the default is the direction of 90 degrees.

Let's say we turned counterclockwise 15 degrees: we would go to 75 degrees. The go to blocks move the Sprite instantly (in a snap) to a chosen grid position. It can go to specific coordinates or other sprites or a random position depending on your program. And the glide blocks are similar to the go to blocks, but the movement is not instant. The Sprite glides. That's why it's called the glide block - it glides across the grid instead of instantly repositioning.

Project Intro 2 - Slithering Snake



1. **Select** the snake sprite
2. **Drag** the "**when {green flag} clicked**" event block into the code area
3. **Pick a starting position** on the grid, and **move there** using a "**go to**" block
4. Use the "**turn**" and "**move**" blocks to replicate a **snake slithering**
   a. ***Ex:*** *[turn {clockwise} 30 degrees], [move 20 steps], [turn {counter-clockwise} 60 degrees]*
   b. ***Repeat***

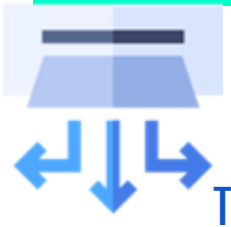


5. **Click** the **green flag**
   a. *Does it work as **expected**?*

Slide 8
In this video, we're going to use those motion blocks in Project Intro 2 - Slithering snake.
Let's get into it. Our first step is to select the snake Sprite and delete the cat Sprite that is already by default there. Second, we're going to drag the 'when green flag clicked' event block into the code area. Third, we're going to pick a starting position on the grid and move there using a go to block. For number four, we use the turn and move blocks to replicate a snake slithering. An example: turn clockwise 30 degrees, move 20 steps, turn counterclockwise 60 degrees, and then move 20 steps and repeat. Finally, click the green flag. Does it work as expected? This is the part of the video where you can pause and try it by yourself, or you can participate at the same time I do it as well, or you can just simply watch. Let's do this.

Now that we're in the scratch environment, Let's go ahead and delete the default cat. We're going to add the snake. I'm going to use the search bar for this. We finally have the snake. That is the first step. Second, we're going to drag the "when green flag clicked" block into the code. I'll blow this up so that it easier to read. We're going to pick a starting position on the grid and move there using a go to block. How does that work? Let's go to 'motion' where the go to block exists. Then we're going to come over to the snake. If you notice underneath the snake, it'll say 'Sprite: snake', and then it will show x and y. Right now it's saying x is negative 69.34. That's showing the position of the snake in the grid area. If I grab this snake and move it, the coordinates update. Now we're at negative 1.44 for x and y is 33. What's really cool is if you notice over where the block selection area is, when you move the Sprite, the coordinates automatically update there as well!

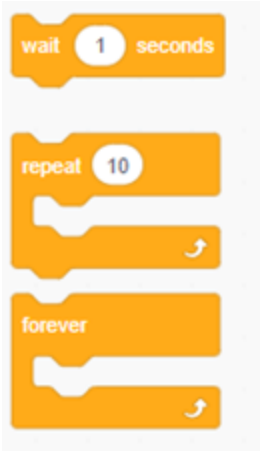
Programming your Sprite - CONTROL!



Control blocks "control" the flow of the progra

The **wait block pauses the program** for the specified **duration**

We use it to **trace output** in steps - ***computers are very fast!***



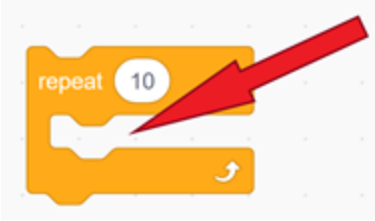
The **"repeat" block** (or **loop block**) is distinct from blocks we've seen

Blocks **attach inside or** on the **bottom** of this block

Code **blocks inside** are **repeated** the given **number of times**

When **finished "looping"**, the **bottom code** runs

The **"forever" block** acts the **same** as the **repeat block**, but **doesn't stop** looping/repeating



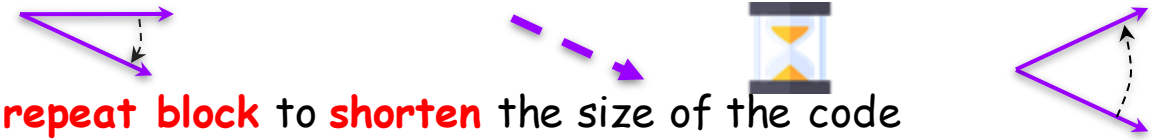
Slide 9
Control blocks control the flow of the program: how fast it happens, where it's going, etc. As we learn more about control (and eventually, decision statements), there will be multiple paths for a program to take. That will affect the flow of the program. For now, we're just going to worry about speed. The wait block causes the program to pause for the specified duration: if it's 'five seconds', it will wait five seconds. If it's '0.5 seconds', it will wait half a second. We use it to trace output in steps, since computers are very, very fast.

The repeat block (or loop block, as I will call it), is distinct from the other blocks we've seen so far, because the blocks attach inside or on the bottom of this block and the code blocks inside are repeated the given number of times. When finished looping or repeating, the bottom code then runs. Anything you would attach underneath of the repeat block, as you can see down here, will execute after the code that is inside of the loop block has run. I call the loop block the "alligator's mouth" sometimes because it kind of looks like an alligator (minus the teeth). It will execute all the code inside of the alligator's mouth before moving on to the rest of the program or script.

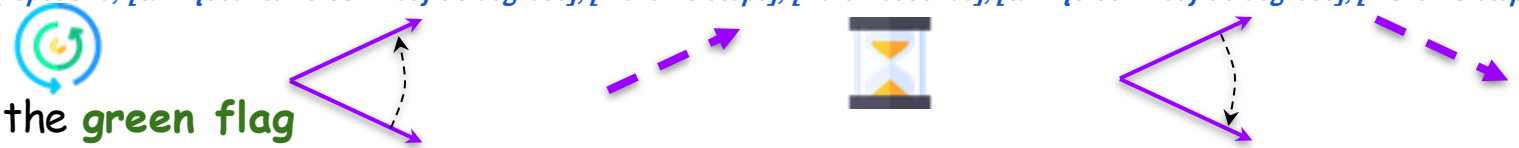
The forever block acts the same as the repeat block but doesn't stop repeating the loop. This is why, as you can see, it has no divot on the bottom in order for a code block to attach to it, because it is going to be the end of the program if it stops moving.

Project Intro 3 - Fixing the Slithering Snake

1. **Select** the snake sprite
2. **Reset** the **direction** of the sprite to **90**
3. **Insert** "**wait**" blocks into the program to **slow the flow** of the program
   - ***Ex:*** *[turn {clockwise} 30 degrees], [wait 1 seconds], [move 20 steps], [wait 1 seconds], [turn {counter-clockwise} 60 degrees]*



4. Use the **repeat block** to **shorten** the size of the code
   - ***Ex:*** *[repeat 10, [turn {counter-clockwise} 60 degrees], [move 20 steps], [wait 1 seconds], [turn {clockwise} 60 degrees], [move 20 steps]*



5. **Click** the **green flag**
   - *Does it work as **expected**?*

Slide 10
Hello, everyone, and welcome to Project Intro 3: Fixing the Slithering Snake. We're going to go through our steps. Our first step, again, is to select the snake Sprite. What I would recommend doing is using the script (program) that we made in the last project video, since this is just updating it to fix it.

We then reset the direction of the Sprite to 90 degrees, just in case we didn't leave it off at 90. At the end, we're going to insert wait blocks into the program to slow the flow.

In this example, we're going to turn clockwise 30 degrees, wait one second, move 20 steps, wait one second, and then turn counterclockwise 60 degrees, or (as we were doing) 15 degrees and 30 degrees. You can wait less than a second, you can wait several seconds - it simply depends on how you want to flow. We'll then use the repeat block to shorten the size of the code. If we wanted our snake to keep going in the last project video, we would have had to take our program and then just copy it over and over, making that script longer and longer! Instead of doing that, we're going to use the repeat block. That way, we don't have to copy and paste the same bit of code over and over and over. And that helps with a little word called redundancy, which basically means that we're not wasting our time doing the same thing.

Again, an example, use the same code as last time, but 'repeat 10' (remember our alligator mouth), put that code inside of there, and then click the green flag. Does it work as expected? With that, this is the part of the video where you can pause and attempted to do it by yourself, or if you'd like, you can do it at the same time as me. Or you can just watch. Let's get into that.

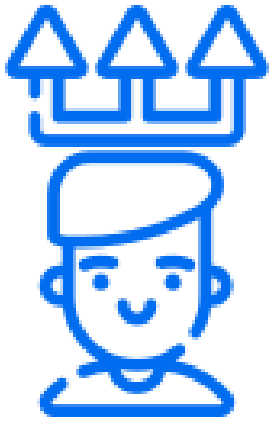
We left off where we finished running our program, but now our snake doesn't move! If we move it, it goes immediately to that position because it’s working through the code so incredibly fast. The first step that really changes what's going on here is to go to control, where we're going to insert wait blocks. In order to see our snake move, we're going to go to wait and go to our starting position, which is what we selected here with these coordinates, and then we're going to wait one second (we can change that to less to make the flow a little different).

Then we're going to turn 15 degrees and move; I want those both to happen relatively fast, so I won't put a wait block in between. That way, we don't have to watch it turn and then move (unless we wanted to). Then we'll have to turn and move again, and then wait again. This repeats several times, with the Sprite turning back to the 90 degree position/direction. Let's click and see what happens. Hey, that kind of started to look like a slithering snake!

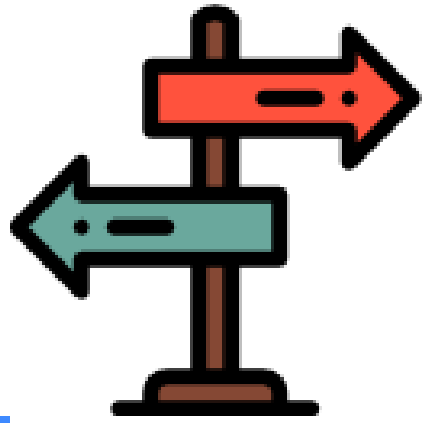
Without the loop, we would pull this out, right click it, say 'duplicate', scroll down, and attach it as many times as should be repeated. But this will be a really long piece of code. I have to Scroll down to find the whole thing. Now, our Sprite kind of moves like a slithering snake. But again, it doesn't go all the way across the screen. And so I'd have to keep duplicating, making this longer and longer, which is not what we want to do. Let's solve that problem, yet leave this extra script here. We're back to what our newer, better working version is, but only one set.

Now we can go in our control area. We can find the repeat block, but we have to decide where to insert it. It will cap out at the bottom of the program as I put all of the code inside the alligators mouth, as you can see here. Now that we put all this inside the repeat block, let’s see how it works. The does kind of look like a slithering snake, but I don't like the way that our Sprite is constantly tilting this way. For this, there is an option that we can select. If we hover over this, it says 'all around' - if we Hover over this other one, it says 'left to right'. It will lock our Sprite facing 90 degrees in one way or 180 the other way (or negative 90). If I pointed it, it goes over this half, it turns around, or if it comes over here, it faces this direction. And that prevents the spinning motion that would be what we just saw. Now, I'll return this to 90 and click the green flag, Let's see how it looks. That looks like a slithering snake to me. That concludes Project Intro 3, which also concludes our introduction to Scratch.

Let's Learn Scratch - Part 2



DECISION BLOCKS in SCRATCH

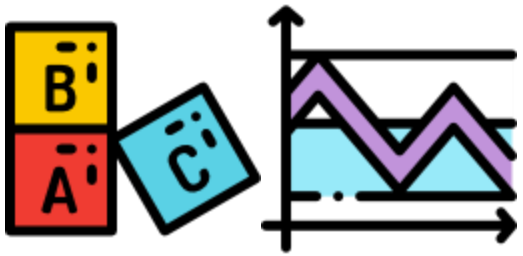


Slide 1
Hello, everyone! Welcome to the ARCS Microcredentials Algorithms and Programming - Decision Blocks in Scratch!

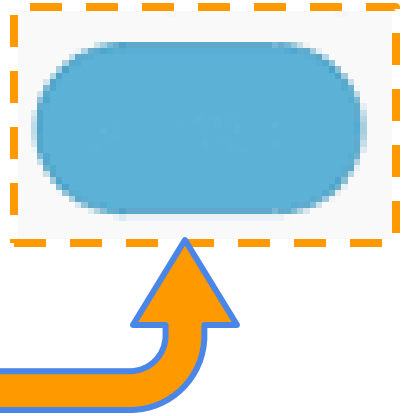
Overview

1. Basic Variable Blocks
2. Boolean Blocks
3. Sensing Blocks
4. Decision Project 1
5. Decision Blocks
6. Decision Project 2
7. Operator Blocks
8. Decision Project 3

Slide 2
In this section, we are going to go over basic variable blocks. We're going to talk about what boolean blocks are. Then we're going to talk about sensing blocks. Then we have a project on decisions. Then we have decision blocks - then we have another project, then operator blocks, then the final Decision project.

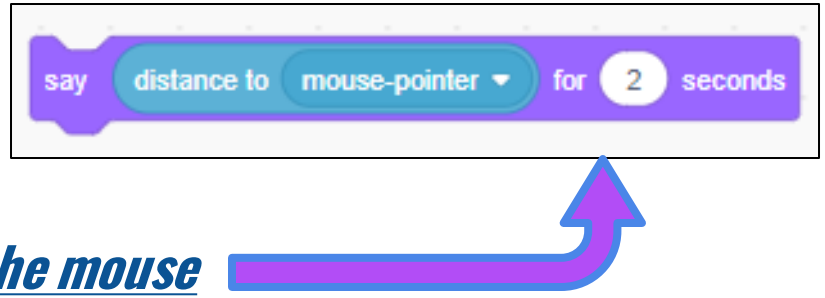


Basic Variable Blocks



- A variable represents a value that may **change**
  - ***However***, the name labeling the variable ***always*** remains the same
  - ***In Scratch,*** blocks with rounded edges are VARIABLE blocks

- Variable blocks hold both `numeric` and `text` values



- Variables can be **inserted** into other blocks
  - ***This example prints the distance from the sprite to the mouse***

Slide 3
Let's go ahead and start talking about basic variable blocks. A variable represents a value that may change.
For example, your bank account balance is technically a variable, because the amount of money in your bank balance may change, or how many students your class has is a variable, because they might change how many students you have.

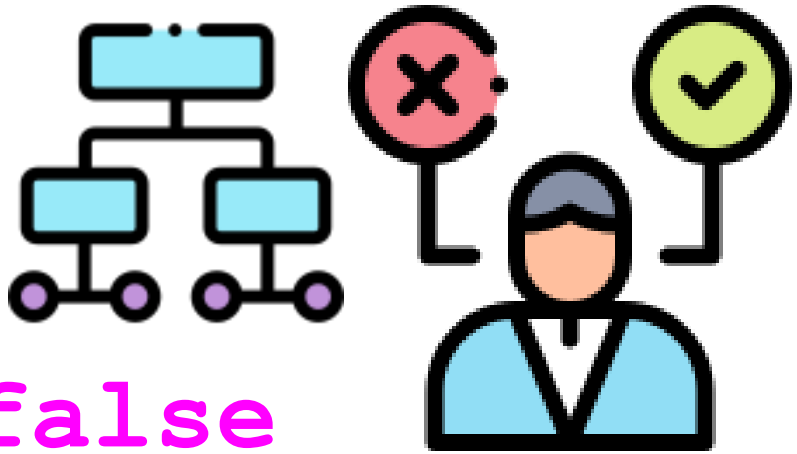
So, however, the name labeling the variable always remains the same. For example "students.
"MyStudents" doesn't change, but the number of "MyStudents" may change.

In Scratch, blocks with rounded edges are variable blocks. Variable blocks can hold both numeric and text values, but they won't hold them at the same time. So, it will either hold a numeric value OR a text value.

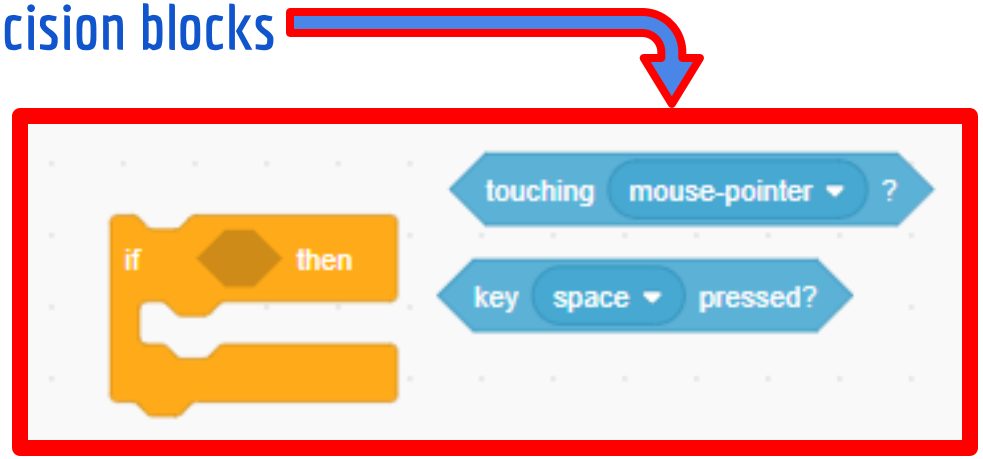
Variables can be inserted into other blocks. We'll see this again here, soon. Here we'll see the example; as I said earlier, in Scratch, blocks with rounded edges are variable blocks, so this is what a variable block would look like - this is how its edges sit.

And as I also said, you can insert them into other blocks, so this example here will print the distance of the sprite to the mouse. This particular variable will hold a numeric value of how many grid points the mouse pointer is away from whichever sprite this code block belongs to.

Boolean BLOCKS



- A boolean represents a value of either **`true`** or **`false`**
  - ***Boolean blocks*** are shaped as **`horizontally stretched`** hexagons
- Booleans are used for **decisions**
  - They can be inserted into the hexagonal spaces in decision blocks

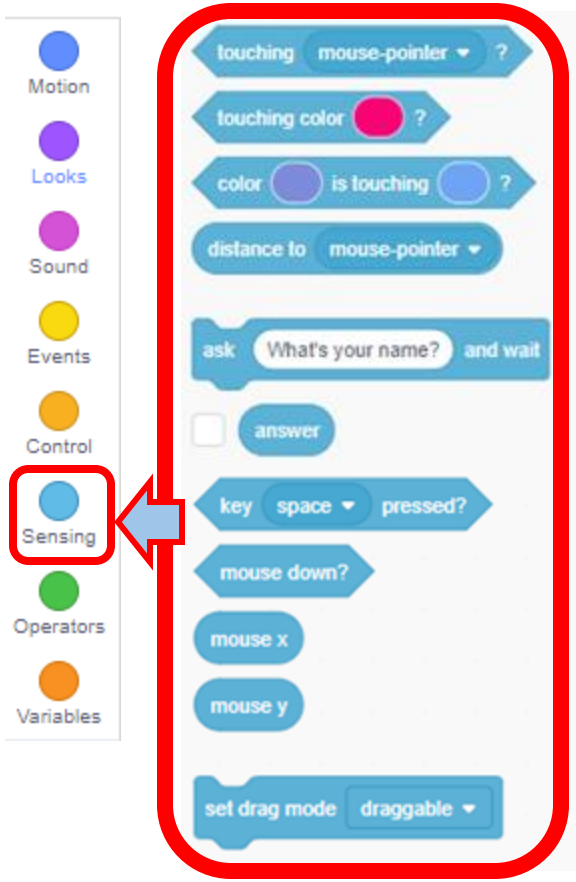


Slide 4
Up next, we have boolean blocks. So, what's a boolean? A boolean represents a value of either true or false, which makes it very simple. There's usually a question or a condition.
For example, is this towel wet? True or false? There you go. That's kind of how a boolean works, and boolean blocks just hold that value.
Boolean blocks in Scratch are shaped as horizontally-stretched hexagons. Here, in a second, we will see an example of that.

Booleans are used for decisions. Should I dry myself off? Am I wet? True or false? That's an example of that. They can be inserted into hexagonal spaces in decision blocks. Or, they can be inserted into round spaces, where variables are, but they might not always work as expected.

(if) These are boolean blocks. As you can see, they have the hexagonal shape and they're horizontally stretched, and they will fit into this hexagonal opening here.
They will also fit, for example, into a say block, which has rounded edges. The say block will then display in the speech bubble what value that boolean block was holding, whether it had been true or false.

Sensing BLOCKS



- Sensing has **both variable and boolean** blocks
- **"Touching mouse"** and **"touching color"** refer to the **sprite**

  - **color {color} is touching {color}"**: in the **sprite area**
  - **"distance to {mouse-pointer}"**: gives **distance to an object**
    - ✓ Default is **mouse pointer**, but *can be a sprite*
  - **The "ask" block**: **says the text** then **expects input** (text box)
    - ✓ Data entered is held by the **"answer" block**
  - **"Mouse down"**: senses if you **clicked** in the **sprite area**
  - **"Set drag mode"**: **choose** if you can **drag** the **sprite**

Slide 5
We will now discuss sensing blocks, from the "Sensing" section of the Scratch code blocks. Sensing has both variable blocks and boolean blocks.

"Touching mouse" and "touching color" refer to the sprite, so is the sprite touching the mouse pointer, and is this sprite touching the specified color. You can tell these are boolean blocks based on the shape.

"Color [color] is touching [color]" is in the sprite area. So, if another sprite that has a specific color is touching a spot in the backdrop that has a different color (the other color), then this will return true. Otherwise, false.

The "distance to [mouse pointer]" gives distance to an object. For example, it could be the mouse pointer, or it could be a sprite. If there's another sprite in the stage or sprite area, then you can get the coordinate distance from one sprite to the other.

The "ask" block says the text then expects an input. It presents a small text box on the screen. Whatever data is entered into that text box is going to be held by the "answer" block. That variable will change depending on what the answer is.

The "mouse down" senses if you clicked within the sprite area. Imagine the mouse is pressing down when you click on the sprite area or the stage. It will return true while it's being clicked.
"Set drag mode" will choose if you can drag the sprite when you are running the program in the full stage, outside of the editor.
And with that, it is now time to begin Decision Making Project 1 - Sensing Experimentation!

and it does. Now, if I change this color to red, it should say false. Now that we finished setting both of those boolean blocks, we're going to choose the "when space key pressed" event block. We then attach a say block just like we did earlier.

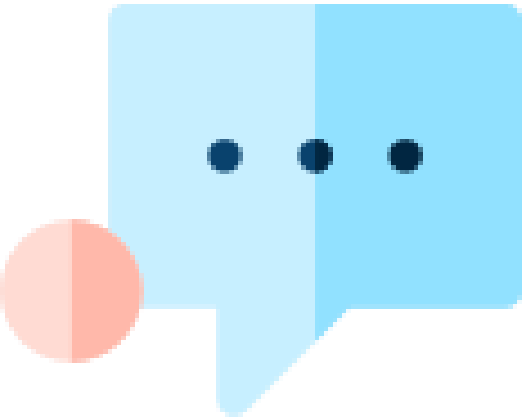
Then what we're going to do is go to sensing and to select the "mouse down" event block, as we're gonna test this boolean block. If you remember our "mouse down" is when you're clicking on the stage area and it's sensing for that click. If you hold down the click, it will stay true.

When you release the click, it will return to false. So, for example, I'm not clicking right now, but if I click the space bar, it will say false. Now, if I press down on my mouse and click the space bar, it should say true. Now, if I pull my mouse away from the stage or Sprite area (I will hold down my click and then click space), it will still say false. Alright, and that concludes decision making Project 1 - Sensing Experimentation.

Decision Making Project 1

- Sensing Experimentation -

1. **Choose** the **green flag clicked event** block
2. **Attach** a **say** block
3. **Select** either the **key pressed** block or the **touching color** block
   - If using the **touching color** block, set the color to ***white***
4. **Test** both **boolean** blocks
5. **Choose** the **when space key pressed event** block
6. **Attach** a **say** block
7. **Select** the **mouse down** block
8. **Test** the **boolean** block



Slide 6
Let's get started with the decision making Project 1: Sensing Experimentation. We're going to first choose the "when green flag clicked" event block, then attach a say block, and select either the key press block or the touching color block. If using the touching color block, set the color to 'white'. We're going to test both boolean blocks, but you get to select one or the other. Choose the "when space key pressed" event block, attach a say block, and then select the "mouse down" block. We're going to test that boolean block as well. This is the part of the video where you can go ahead and pause if you'd like to go ahead and try it yourself without my help, or if you'd like to work it at the same time, that is fine as well. Or if you just want to watch, that's fine. Let's get into it.

I'm going to use a different Sprite for this, just for the fun of it. I'm gonna go with this beautiful butterfly. Our first step is to choose the "when green flag clicked" event block, and I will blow this up so you guys can see what's going on.

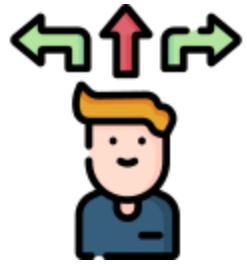
Next we're going to attach a say block, and then I'm going to get the "key press" block right down here. We'll go ahead and stick with space. As I said before, you can put booleans inside of round variable spots. Next, I'm going to hold down the space bar and then press the green flag. We should see that it says true. Now I'm going to let go of the space bar and press it again. It should say false, which it did.

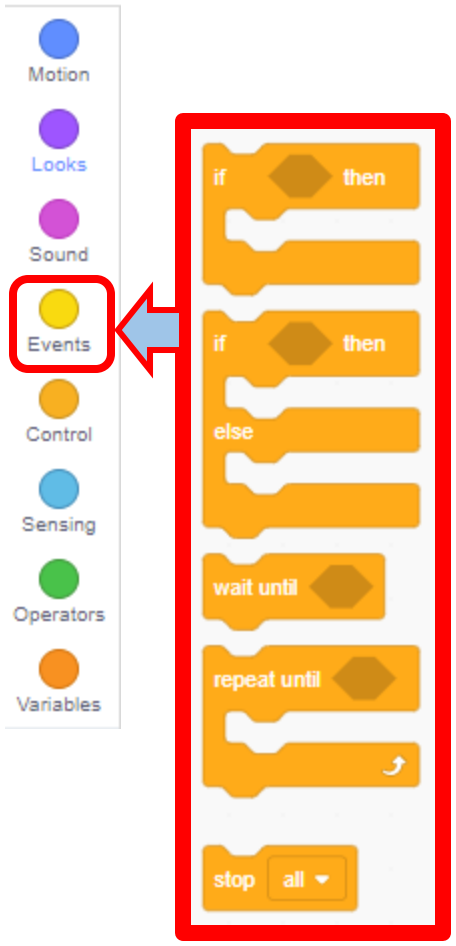
After that, we're going to test the touching color block. I'm going to pull out this key press block and get rid of it and grab the touching color block, then set it to white. If I click the green flag, it should say true, and it does. Now, if I change this color to red, it should say false.

Now that we finished setting both of those boolean blocks, we're going to choose the "when space key pressed" event block. We then attach a say block just like we did earlier. Then what we're going to do is go to sensing and to select the "mouse down" event block, as we're gonna test this boolean block. If you remember our "mouse down" is when you're clicking on the stage area and it's sensing for that click. If you hold down the click, it will stay true.

When you release the click, it will return to false. So, for example, I'm not clicking right now, but if I click the space bar, it will say false. Now, if I press down on my mouse and click the space bar, it should say true. Now, if I pull my mouse away from the stage or Sprite area (I will hold down my click and then click space), it will still say false. Alright, and that concludes decision making Project 1 - Sensing Experimentation.

Decision (Control) BLOCKS





Decision blocks control the flow of the program (algorithm)

The program can act differently depending on *varying conditions*

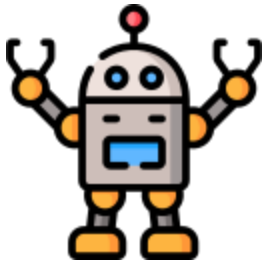
The **if block** runs the code inside if the **boolean block** it has holds a value of **true**

Otherwise, it **skips** it and continues to run the **following** code

The **if – else block** works like a normal **if block**

However, if the boolean is **false**, it will **only** run the code within the **else**

Wait until pauses the program until the boolean block is **true**

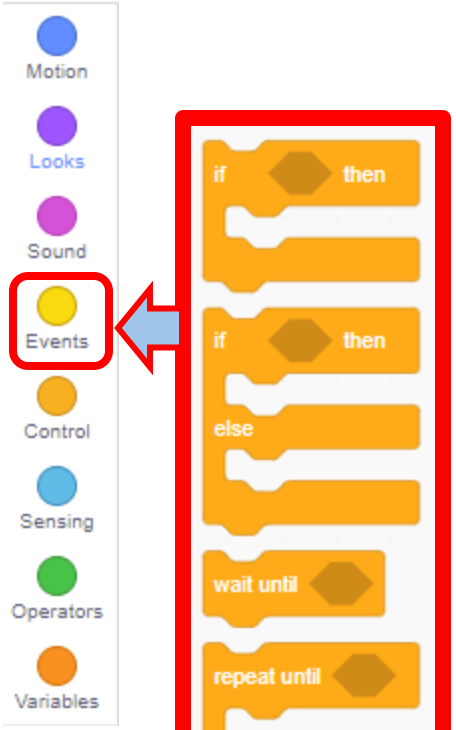


Slide 7
Decision blocks control the flow of the program or algorithm. The program can act differently depending on varying conditions. There seems to be a small error in our slides here where it should be over control instead of events, as the if then else and wait until and repeat until and stop blocks all exist inside of control.

The if block runs the code inside of the alligators mouth for the if statement (or if block) if the boolean block it holds has a value of true. An example of that would be if touching color, where if our sensing block for touching color came back true, then it would execute what's inside of the alligators mouth here. Otherwise, it will skip and continue to run the following code that you would attach underneath of the if statement, or if block.

The if else block works like a normal if block. However, if the boolean is false, it will only run the code within the else. Here's how this works: the first half of the if block will run if this is true; otherwise, or else, it will run what’s inside the second alligators mouth set. The "wait until" pauses the program until the boolean block that it holds returns true or has a value of true.

Decision (Control) BLOCKS {Continued}



The **repeat until** block **repeats** code inside while the boolean is **false**

Once it is **true,** it will **stop** executing the **loop**

The **stop** block has a few options

- **All**
  - **all scripts** that may be running
- **This script**
  - **only** the script the **stop** block is **attached** to
- **Other scripts in sprite**
  - **all but** the script the **stop** block is **attached** to

Slide 8
The next box we will be looking at is the repeat until block, which is different than every other decision block that we've seen so far, because the repeat until block repeats the code inside of its alligator's mouth while the boolean is false. So for all of the other blocks so far, they have been needing a true value in order to execute the first thing. If then else requires a true for the first branch to execute.

The wait until requires a true before it moves on down the code, but the repeat until requires a true before it stops repeating the code inside of its alligators mouth. Once it's true, it will stop executing the loop. The stop block has a few options. As you can see, there's a little arrow sitting here next to the stop block, which give it a few options. There is the all option and all scripts that may be running in your program over multiple Sprites will stop executing if you have stop all.

"This script" stops only the script that the stop block is attached to. For example, if I drop this at the very end of a program and I said stop this script, it would stop it. Or if I put it in an if statement, it would stop it if that if statement was true. Otherwise it would just keep executing the code. Lastly, there's other scripts in Sprite which is all the scripts in that Sprite stop except for the one the stop block is attached to. And, as you can see here, this is what it looks like.

Decision Making Project 2

- Simon Says -

1. **Choose** a sprite to represent "Simon"
2. **Choose** an **event** block as your start
3. **Choose** a command from **a key pressed**, **touching mouse pointer**, or **mouse down**
4. **Say** "**Simon says {do action}**"
5. **Wait** until action is completed
6. **Say** "**{do a different action}**"

7. **Repeat** an **if** statement `50` times that checks if the action is **completed**
   - **Add** a wait block for `0.1` seconds
   - **If** the action is completed **say** "**oops gotcha**"

8. **If** the repeat block finishes executing **say** "**You win!**"

Slide 9

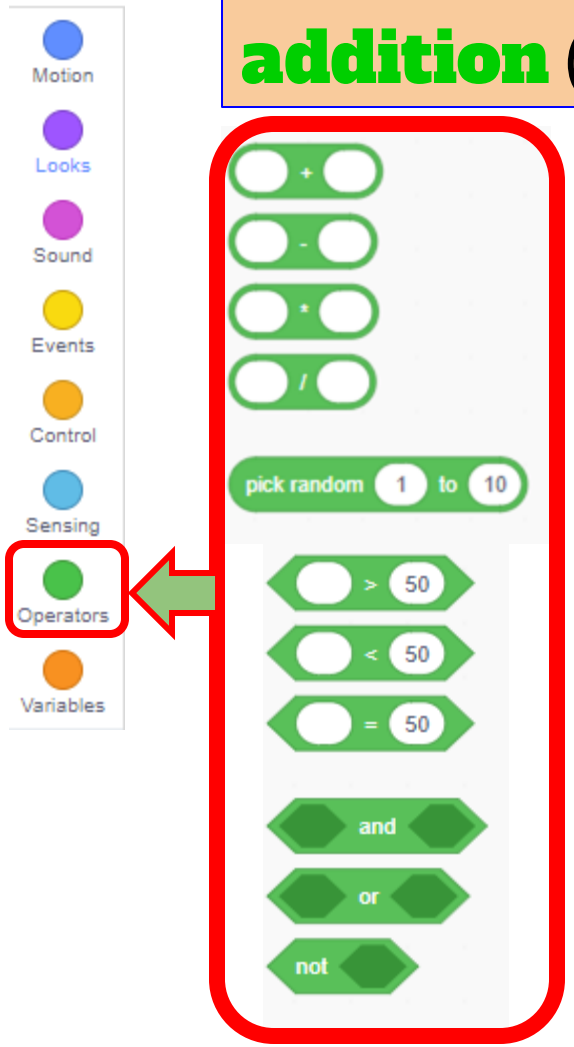
Let's get started on this decision making Project - 2 Simon Says. Our first step is to choose a Sprite to represent Simon. Second, we're going to choose an event block as our start - it could be whatever you want, whether it's a when space key pressed or a when green flag clicked block. I would recommend when green flag clicked, choose a command from a key press, touching mouse pointer, or mouse down. Say, "Simon says, do [action]" - whichever action you chose from the previous step. It can tell the user to press a specific key or hover over him or mouse down (which would be click on the screen). Have him wait until the action is completed, and then say, "do a [different action]", but without the Simon says this time. And repeat an if statement 50 times to check if the action is completed. You're going to want to add a wait block for 0.1 seconds. That's five seconds of waiting. If the action is completed, say "Oops, gotcha". If the repeat block finishes executing, say, "you win".

Alright, so this is the part of the video where you can pause and attempt to do it by yourself, or you can wait and we can do it together. Or you could simply just launch. Let's go ahead and get into it. Our first step is to choose the Sprite to represent Simon. I'm going to delete Sprite number one, this default cat, and find something that I want to talk to us. Let's go with this dress; this dress is Simon. Up next, we're going to choose an event as our start. So, like I said, I recommend the when green flag clicked event block. Then let's choose a command from a key pressed, touching mouse pointer, or mouse down. I'm going to do a key pressed. We will sense if a specific key is pressed. I'm going to go with W for win. We're going to go to say block. And then we're going to say "Simon says, press the w". Then we're going to use a wait until block, then do key w pressed.

Now we're going to do a different action. Out of the three actions that we haven't done, we're going to say, "click on the stage". And now what we're going to do is get a repeat block from control to repeat 50 times. I'm going to add a wait block for 0.1 seconds, then put an if statement and say sensing mouse down. In there, it says "Oops gotcha". If they don't click it and they wait for five seconds, we say "you win". Let's see if this works. I Press W. It says, now click on the stage... "Oops gotcha". Whoops - I messed up! It then says "you win". It's not supposed to do that. And Here's how to fix that. So after this if statement executes, it continues to repeat, even though we pressed our mouse down. So what we need to do is go back to control to grab the stop block. And we can say stop "this script". Now we can press the green flag and see if it works. "Simon says, press the W key." It's pressed. "Click on the stage." Clicked. "Oops gotcha." Now does it say you win? No, it doesn't. And you can tell because the code is no longer highlighted in yellow. So if we press our W key and then we don't click on the stage, it should say in about five seconds, "you win". That was Project 2.

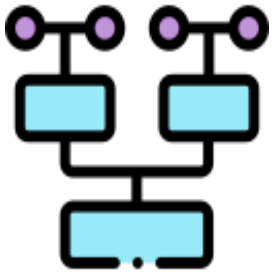
Operator BLOCKS

These blocks allow for operation on data, including addition (+), subtraction (-), multiplication (*), and division (/)



Less familiar operations:

+ Pick a random number within a given range
+ Greater-than, less-than, and equal-to blocks take two variable blocks and compare values

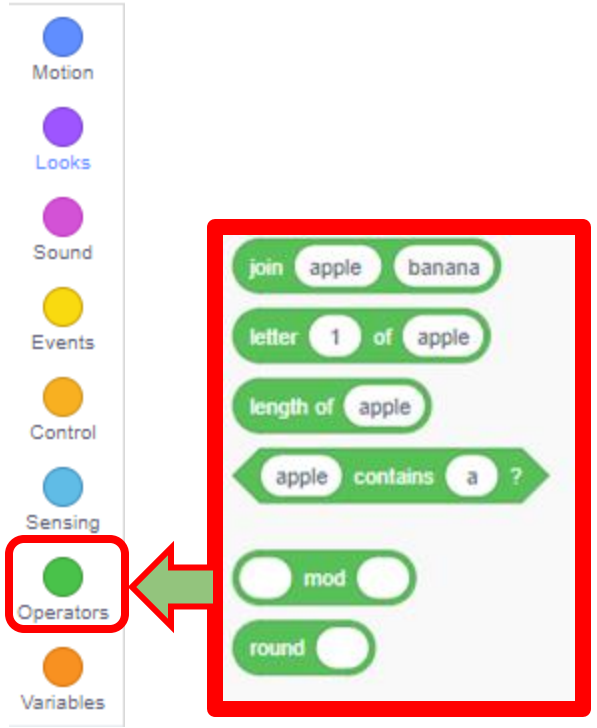


+ The "and" block takes two boolean blocks
  - *Both must be true* for it to be true
+ The "or" block also takes two boolean blocks
  - *Only one needs to be true* for it to be true
+ The "not" block is opposite of the input boolean
  - *Ex:* a true block will result in false

Slide 10
Operator blocks are pretty complicated because these blocks allow for operation on data, which includes addition, subtraction, multiplication, and division. And here are some less familiar operations. So there's a pick a random number within a given range. We can pick a random number from one to 10 or five to 500 or 500 to a thousand. And these are these blocks here. So this is the greater than sign, or the greater than block.

This is the less than block, and this is the equal block. This is the and block. For example, key pressed and touching color would both have to be true in order for this whole block to be true. Or we could say 51 is greater than 50 and 50 equals 50. Both of those are true statements. This whole block would be true. With the or statement, we could say 51 is greater than 50, but 49 equals 50. But because 51 is greater than 50, this whole block would return true because it only needs one to be true. For the and block, true and true equals true, but true and false equals false. The not block does the opposite. So not true equals false and not false equals true.

Operator BLOCKS {Continued}



+ "join" combines two text values
  - *Ex:* "apple" & "banana" = "applebanana"

+ "letter of" gives the letter at the chosen position in text
  - *Ex:* "letter 2 of apple" is "p"

+ "length of" gives the number of characters in the text
  - *Ex:* "apple" = 5

+ "mod" is short for modulo
  - It takes two numbers and gives the remainder after division

+ "round" rounds numbers to the nearest ones place

Slide 11
Let's get into even more operator blocks. These are just to learn; most of you probably won’t use these very much. The join block, which is right here, combines two text values together. For example, "apple" and "banana" equal "applebanana" smashed together.

The 'letter of' give the letter at the chosen position in the text. In this example here, letter two of Apple is P, and then the length of gives the number of characters in the text. So, length of Apple is five. And then our last one is mod, which is short for modulo. Modulo sounds complicated, but it's really not. It just takes two numbers and gives the remainder after the division. Let's say how many times does three go into seven? Instead of returning the two, it would return 1 because 3 goes into 7 twice. Last, but certainly not least, round block rounds the numbers to the nearest ones place. So if there were any decimal points, then it would take that number from the decimal point and make it a whole number.

Decision Making Project 3

- Interview -

1. **Select** a sprite to be our **interviewer** 
2. **Chose** the **green flag clicked** **event** block
3. **Use** the **ask** block to ask the user their **name** 
4. **Using** the **join** block, say "**Hello {answer}**" 

5. **Use** the **ask** block to ask the user their **age** 
6. **Either** **subtract** the **age** from the **current year** to find **birth year**...

   ...**OR** **see** if age is **over** the age of **18**

7. **Say** the **birth year** or if they are an **adult** or not 

Slide 12
Welcome to decision making Project 3 - Interviewer. We're going to select a Sprite to be our interviewer, and then we're going to choose the when green flag clicked event block. Next, we're going to use the ask block to ask the user their name.

After that, we're going to use the ask block again to see their age. And we're going to either subtract the age from the current year to find the birth year or we're going to see if the age is over 18. So we're going to say the birth year or if they are an adult or not. Let's go ahead and get into this. Alright.

For the first step, let's select the new Sprite for our interviewer. I'm going to choose the Cheesy Puffs as our interviewer. Now we're going to do the when green flag clicked event block. We're going to use a new block that we haven't used yet: the "ask [question] and wait" block. And we're going to ask about people's names, so we don't have to change the thing. Now, we're going to use a join block. Let's get the join block.

Then, we are going to get a say block, and then go to our sensing to get the answer block. So now we can join together "Hello" and the answer. Now it'll say Hello answer and make sure you have that answer. Put a space on the end of "Hello"; otherwise it'll mash the two words together. Let's get another ask block and ask about the age this time. First, Let's start with subtracting their age from the current year to find birth year. First, we're going to subtract the age from the current year. So if you go to your sensing and you look towards the very bottom, there is a "current" block and you can select year, month, date, day of week, hour, and minute.

I'm going to choose current year. That is its default. We will drag that over into our code area. Then, we get our operator. I'm going to get subtraction. We're going to subtract age from current year. I'm going to say "you were born in...". Let's see how it works. What's your name? My name is Grayson. Hello, Grayson. What is your age? I am 19. You were born in 2002. It would normally be correct, but I was, in fact, born in 2001 as I was born in December. So there is a time frame there where it will be wrong because it doesn't account for birthdays, but it does get pretty close. It doesn't have to be exact. This is just to get the gist of using operator blocks.

Let's redo this. I keep the answer block, as I don't need the rest. I'll set this up here so we can remember it, but it's not going to do anything. Let's see if they are over the age of 18. So we're going to go to our greater than or less than block, and we're going to say the age is greater than 17. Because if you're 18, you're an adult. Is our answer greater than 17? Let's get an if block. An if then or else block actually makes it easier. Let's say "Congrats. How does it feel to be so old?" For an adult. If they're under the age of 17 or 17, then we will say "Aren't you a little young to be riding a roller coaster?" Let's test it. "What's your name again?" My name is Grayson, but this time I'm going to be Zagornorth. "Hello, Zagornorth. What is your age?" I am 19. "How does it feel to be so old as an adult? "And we can increase the time of that display. So we can put that in four seconds, and we'll put that to four seconds as well. That way, it'd be nice and easy to read. And with that, we conclude decision making Project 3.

Let's Learn Scratch - Part 3

USING and CREATING

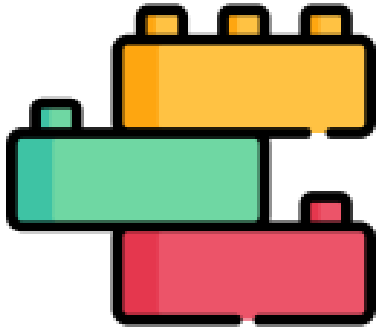
Variables in SCRATCH

Slide 1
Hello, everyone, welcome to the ARCS Microcredentials Algorithms and Programming, and welcome to the section of Using and Creating Variables in Scratch.

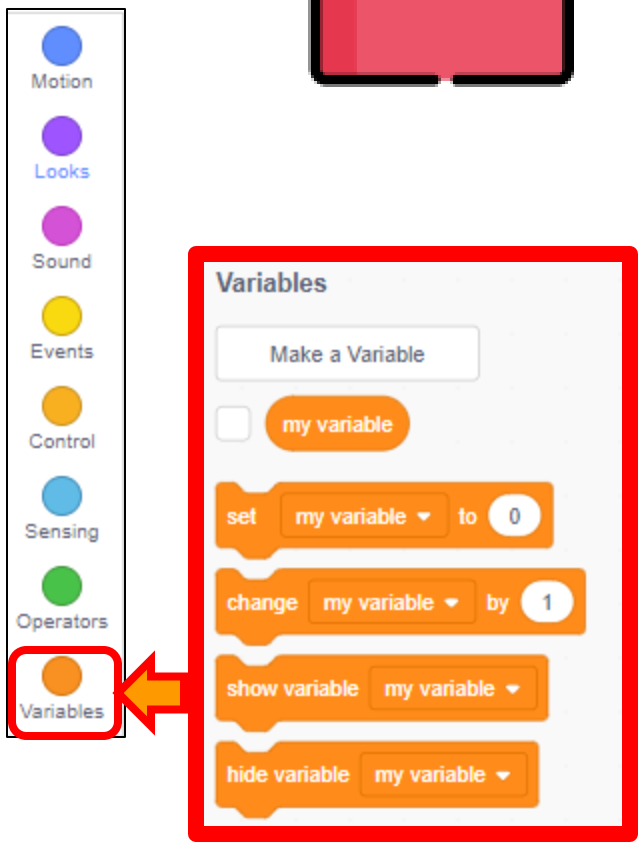
CONTENT

- **Variables** and **Variable Blocks**
- **Options** for **Variable Blocks**
- Variables Project 1

Slide 2
In this section, we're going to be going over Variables and Variable Blocks, Options for Variable Blocks and Variables Project Number One. Let's go ahead and dive into this.



Variables and Variable Blocks



Variable blocks are created to hold data

The "Make a Variable" button allows you to add another variable.

YOU CAN PICK THE NAME OF YOUR VARIABLE!

HOWEVER, it's **best** to name the variable after the **DATA** it holds.

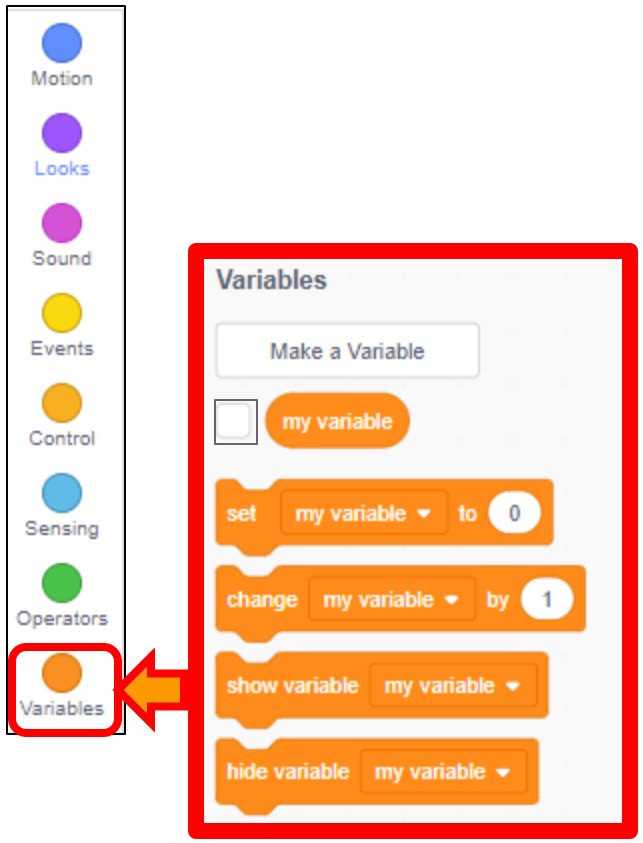
(Ex: the variable block called "*quiz grade*" holds the grade of a quiz)

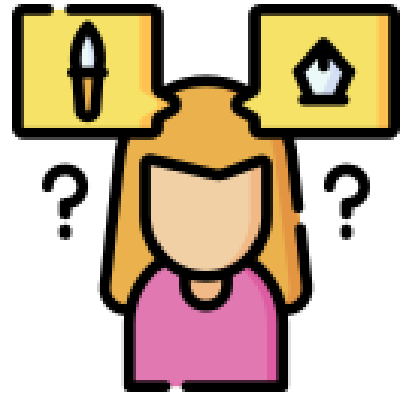
Slide 3
We are going to talk about variables and variable blocks. We've already learned about variable blocks (to an extent) in previous sections, but now we will learn about how to create more variables, including custom variables. As we already know, variable blocks are created to hold data.

The "Make a Variable" button allows you to add another variable. There is a variable section here (it will pop up soon) where we can make a variable. You can pick the name of the variable. However, it is best to name the variable after the type of data it holds, or what kind of data it's holding.

For example, the variable block called "Quiz Grade", holds the grade of a quiz, or "Number of Students" would hold the number of students. Here, as you can see, the variables section is under the operator section. These blocks will help you after you make the variable – we will discuss them further.

Options for Variable Blocks





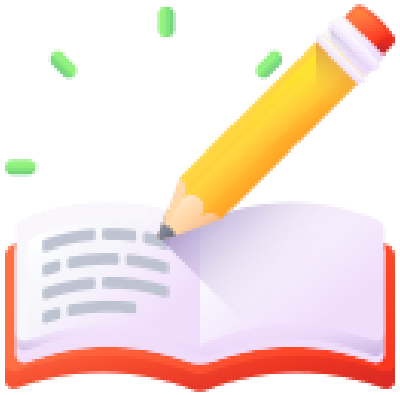
- **Click** the checkbox next to a variable to **show it on the stage**
- **"set {variable name} to"** sets the value held by the specified variable block
- **"change {variable name} by"** adds to the variable's value by the numeric value specified
  - ➢ If the variable is **text** it sets the value to that **specified number**
- **"show variable"** and **"hide variable"** are for **displaying** the selected variable **on the stage**

Slide 4
We've got some options for variable blocks, so we can click the checkbox next to the variable to show it on the stage. That little area where we see our Sprite will display our variable and the value that it holds if we check this box next to the variable. The "set variable name to" sets the value held by the specified variable block. For example, in our previous slide, we had Quiz Grade. If one of the students got a 99 on the test, we set Quiz Grade to 99.

Next is the "change variable name by" block, which adds to the variable's value by the numeric value specified. Let's say our variable holds 95, which means we gave a student in our same example of Quiz Grade a 95 on the test. Let's say they want an 100 on their test, so we gave out an extra credit question worth five points and they got it correct. Well, we could change Quiz Grade by five, which would change it from 95 to 100.

And note: if the variable is a text value, it sets the value to that specified number. For example, if we have a variable that holds a student's name, then we said change variable by five instead of holding the name, it would just wipe away the name and put in the value of five instead.

Show variable and hide variable are for displaying the selected variable on the stage. Instead of having to check and Uncheck that variable here manually, you could during your script show and hide the variables.



Variables Project 1

- HOMEWORK GRADING -

1. **Choose** a sprite
2. **Select** the **when green flag clicked** event
3. **Create** **two** new **variables**, called ***name*** and ***grade***
4. **Ask** for the **name** of the student
5. **Set** the ***name*** **variable** to the answer
6. **Ask** for the students ***grade***
7. **Set** the ***grade*** **variable** to the answer
8. **Say** "**{name} has a grade of {grade}**"

Slide 5
And now it is time for Variables Project 1 - Homework Grading. We will go over this in the next video:
Hello and welcome to Variables Project 1: Homework Grading. Let's work through these steps. First, we're going to choose a Sprite. Second, we're going to Select the when green flag clicked event. And then we're going to create two new variables called name and grade. We're then going to ask for the name of the student and then set the name variable to the answer. After that, we're going to ask for the student's grade and then set the grade variable to the answer.

We're going to display "{name} has a grade of {grade}". Alright, this is the point of the video when you can go ahead and pause and attempt to do this by yourself. Or, you can follow along and do it with me -or simply just watch. Let's do this! First things first, like always, Let's go ahead and delete the default Sprite. We'll choose a new one. Who do we want to store the student's grade? Let's have the Griffin of Good Grades be in charge of storing the grades right there in the middle. For the second step, let's and choose when the green flag is clicked.

We will go to the variable section for step 3 and make two new variables, the first one being name. And as you can see here, the check is marked. And if you look at the stage, you can see that it displays. I'm going to go ahead and uncheck that, and make another variable called grade. Go ahead and uncheck that as well.
Now, we're going to go to sensing to ask what the student's name is. We're going to add the answer; with that, we're going to go to our variables to set name to answer. Then we're going to ask for the students grade. Let's go to sensing again, get another ask block and another answer block to the variables. Let me add my apostrophes so I have proper grammar. Then, I'm going to set grade to answer.

And then we're going to need a join here. I actually have to get two joins – it's getting complicated.
Let's test it by clicking the green flag. What is the student's name? Bartholomew. What is the student's grade? Bartholomew is doing really well. He's got a 95 in the class. The Good Grace Griffin will tell us the student's name and grade. I'm going to make it five seconds just so you can really see that when it's finished. We're going to talk about Charlie now. Unfortunately for Charlie, he's not doing so hot - I mean, he's not doing bad. He's just got a 89. And that concludes the Variables Project 1: Homework Grading.

APLR Module 1 Assessment

There is no quiz for this module.

Congratulations, you have completed this module! **Click here to return to Course Materials (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)** .

Algorithms and Programming Module 2

This learning module will focus on **Sequences**.

When you complete this learning module, you should be able to:

- **Construct** a sequence of steps (algorithm) to accomplish a task.
- **Identify** the beginning, middle, and end of a sequence.

ALPR Module 2. Teacher Competencies and Alignment with CS SOLs

This learning module includes two teacher competencies (in bold), both of which are aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 1. Construct a sequence of steps (algorithm) to accomplish a task.

ALPR Teacher Competency 2. Identify the beginning, middle, and end of a sequence.

- CS K.1. The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing, emphasizing the beginning, middle, and end.
- CS 1.1. The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively, including
  a) sequencing (including ordinal numbers) and;
  b) simple loops (patterns and repetition).
- CS 2.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and,
  c) identifying events.
- CS 3.1. The student will construct sets of step-by-step instructions (algorithms), both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and
  c) using events.
- CS 4.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively
  a) using sequencing;
  b) using loops;
  c) using variables to store and process data; and
  d) performing number calculations on variables (e.g., addition, subtraction, multiplication and division).
- CS 5.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively,
  a) using sequencing;
  b) using loops;
  c) using variables to store and process data;
  d) performing number calculations on variables (addition, subtraction, multiplication and division); and
  e) using conditionals (if-statements).

Sequences

View the videos in the order suggested in the file name. A PDF of the slides and a transcript of the videos is also included for accessibility.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**


















In this activity we are going to work with blocks of instructions (preset dances) in order to build an algorithm, or sequence of steps. The “blocks” we will be working with are preset dance moves. For this activity we will be using the “The Moonwalk”, “The Robot”, “The Dougie”, “The Twist”, and “The Floss”.

Attached with these instructions is a document to print off and cut out. You will be creating cards to use to build an algorithm using these prebuilt dances. You will notice there are more than just the cards describing the dances.

Duration of each dance is 5 seconds.

Loop cards: You will have three “Start of loop” cards and three “End of loop” cards. Any other cards you place between these two cards will be repeated 4 times. You can place cards before and after these cards.

Pause cards: When coming across these cards pause the sequence for 5 seconds.

End of dance card: This card is to signal the end of the algorithm.

Remember that once you make the algorithm it is in a sequence you must follow. You don’t have to use all the cards. Your algorithm can be however long or short you choose.

Dance Algorithm Activity

Grade Level: K-5
Duration: 30 minutes

Prior Knowledge: Simple commands and body movements

Introductory Activity: Ask the students if any of them like to dance. Say “What are your favorite dance moves?” Ask the students how they would describe a dance to one of their classmates. “Do you do your favorite dance the same each time or is it different?” Ask if they created their dance or is it a popular dance already?

Main Activity: Ask the students to name a popular dance (examples: Chicken Dance or Hokey Pokey). What steps are required in order to do that dance? As a group, create an ordered list consisting of the steps for your chosen dance.

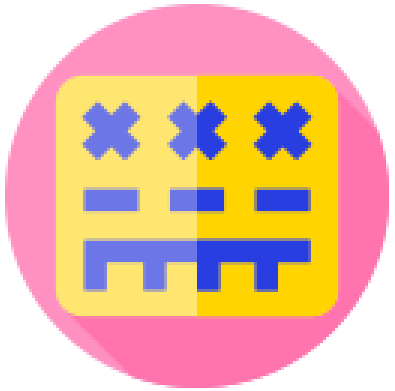
“But what if we wanted to make up our own dance?” Put up a list of five different dance moves (provided examples: Carlton, Floss, Moonwalk, Running Man, Sprinkler). Ask for student demonstrators for each dance. Hand out card packs consisting of two “Wait 5 seconds”, two “Repeat last dance”, and two copies of each dance.

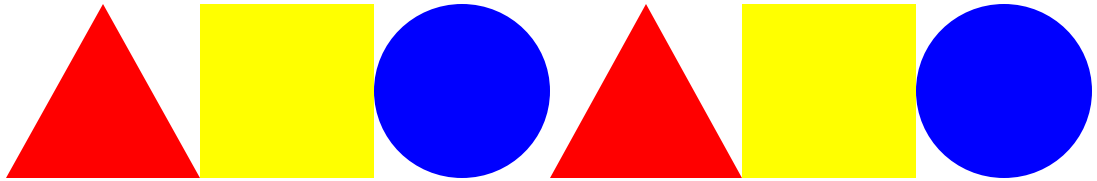
Students arrange 5 cards together. They can compose their dance of whichever moves they would like. Choose a volunteer student to demonstrate their dance. Ask “can anyone else do that same dance?” Have another volunteer try to replicate the dance. Point out any differences in order. “You didn’t follow their steps.” Emphasize that once a sequence or algorithm is defined, that is the order of steps it needs to follow to accomplish the task.

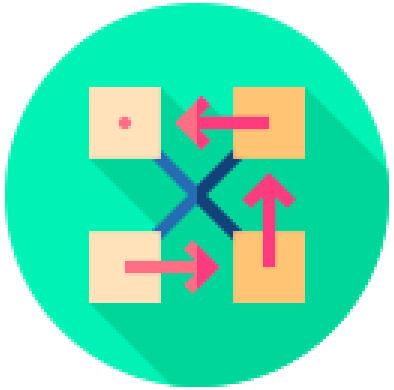
Have students pair up and try to do each other’s dance. They may need to keep the cards on a desk or the floor to refer to.

Wrap-up: Have the students sit back at their desks. Ask for observations about that process. “What was the most fun part?” “What was difficult about doing someone else’s dance?”

Note that many times these sequences are made for fun. They allow us to be creative and share that creation with other people. Sometimes, we are allowed to change our steps for things as we go. Other times those steps get locked into place and we have to use them as they are.







INTRO TO

SEQUENCES

and

ALGORITHMS

WHAT IS

COMPUTATIONAL THINKING?

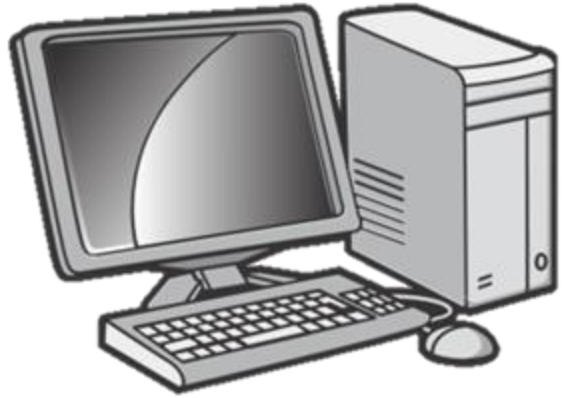
"*The thought processes involved in*

expressing solutions

as

- ***computational steps*** *or* ***algorithms***

that can be carried out ***by a computer.***"



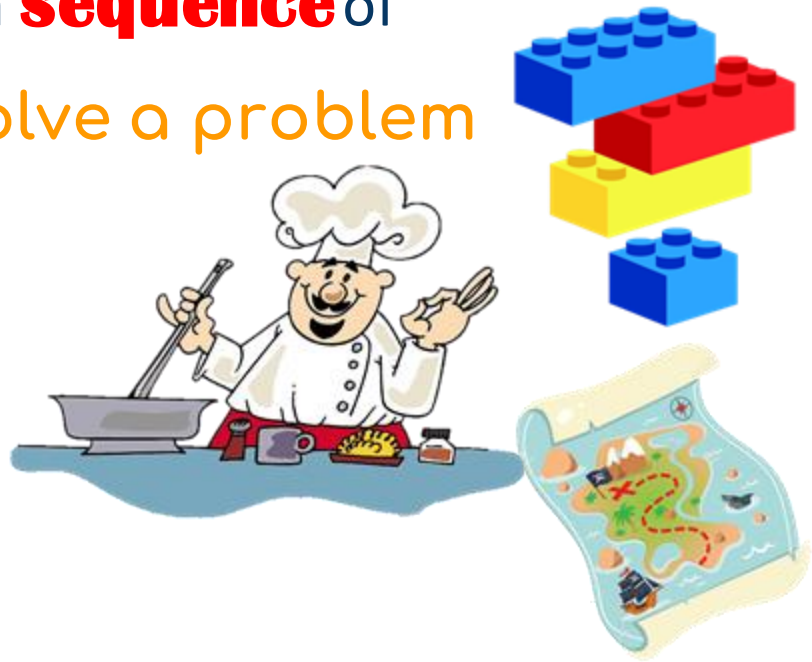
We create **algorithms** to **cause change**.

They are carried out by a **computer** or a **person**.

{After all, the brain is a super computer.}

WHAT IS AN ALGORITHM?

A **sequence** is an **ordered list**. An **algorithm** is a **sequence** of **instructions** and **steps** that solve a problem



Examples of algorithmss in daily life:

⇒Instructions from a **Lego** set

⇒Following any **cooking** recipe

⇒Following directions on a **map**

We use sequences and basic algorithms **all the time** in our daily lives!

HOW TO DEVELOP A BASIC ALGORITHM USING SEQUENCES

Simply take a goal and break down the steps in order to achieve it!

A good example of a goal/problem to solve is grading homework.

LET'S BREAK IT DOWN!

1. **Choose** a student's assignment to grade
2. **Compare** the answers
3. **Calculate** the total grade

We could break those three steps down into **subproblems**!

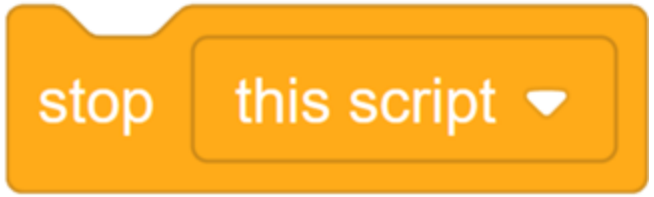
In this example, subproblems include **finding the order** for grading assignments and determining **how many points** each problem counts for, etc.

What are some other subproblems?

Sequences Project – Make an Algorithm

- Your goal is to eat a cake, **but you don't have one!**
- **Develop an algorithm** in order to accomplish your goal

  Break it down into as small steps as possible
- **Assume you already have some of the ingredients you will need**



Breaking Down Our Algorithm

1. **Check** what ingredients we already have
2. **Make a list** of what we need
3. **Get** what we need
4. **Mix** the ingredients
5. **Bake** the cake
6. **Eat** the cake!

Complete the unplugged activity!

Create a “dance” algorithm from the provided dance moves.

Supplemental Information on Sequences

We want to clarify that sequences and algorithms are not the same thing – they aren't synonymous:

- A **sequence** is an **ordered list of ANY ITEMS**
  - A sequence can either be **finite or infinite (it can end, or go on forever!)**
  - This can be an ordered list of **any type of item (not just numbers!)**
  - **Here are some examples:**
    - **Sequence of words:** hot cross buns, hot cross buns, …
    - **Sequence of letters:** A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
    - **Sequence of emojis:** 😀😐😎 😀😐😎 😀😐😎 😀😐😎
    - **Sequence of shapes:** ♥ • ∞ ° O Δ ♥ • ∞ ° O Δ ♥ • ∞ ° O Δ ♥ • ∞ ° O Δ
    - **Sequence of numbers:**
      - 1, 2, 3, 4, 5, ...
      - 1, 3, 5, 7, 9, ...
      - 2, 4, 6, 8, 10, ...
      - 1, 2, 4, 8, 16, ...
      - 1, ½, ¼, 1/8, 1/16, …
      - 1, 1, 2, 3, 5, 8, 13, 21, …
      - 1, 4, 9, 16, 25, …
      - ½, 1/6, 1/12, 1/20, 1/30, ..

- An **algorithm** is an **ordered list of INTRUCTIONS AND STEPS** that **accomplish a task**

  - Algorithms must have some **objective or goal** behind their steps
  - Think of an algorithm as a **sequence of instructions and steps**
  - **Here are some examples:**
    - **Algorithm for calculating the average:**

      1. Count how many numbers you have
      2. Add all of the numbers together
      3. Divide that sum by the count of numbers you had

    - **Algorithm for making a PB & J sandwich:**

      1. Put a slice of bread down on a plate
      2. Spread some peanut butter over the bread

3. Spread some jelly over the peanut butter
4. Put a slice of bread on the top of the jelly

ALPR Module 2. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 2. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of these CS SOL competencies across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Sequences.pdf
(https://canvas.odu.edu/courses/185316/files/44845394/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845394/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 2. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
K.1 The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing, emphasizing the beginning, middle, and end.	English: K.1 (oral communication); K.8 (sequence stories using beginning, middle, and end) Mathematics: K.2 (sequence sets); K.13 (patterns) Science: K.1 (across all standards); K.3 (sequencing objects); K.9 (patterns in nature); K.10 (change) Social Studies: K.1 (sequence events); K.1 (collaborating and participating in classroom activities)
1.1 The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively, including a) sequencing (including ordinal numbers) and; b) simple loops (patterns and repetition).	English: 1.1i (giving simple directions); 1.2b (tell stories in sequential order) Mathematics: 1.2c (placing numbers in order); 1.3 (placing numbers in order); 1.14 (growing and repeating patterns) Science: 1.1b (planning investigations);
2.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and, c) identifying events.	English: 2.1k (give and follow multi-step directions); 2.2a (use the story structure of beginning, middle, and end to tell a story of an experience); 2.10e (organize writing to include a beginning, middle, and end) Mathematics: 2.2 (determining patterns); 2.16 (describe the core of a repeating pattern, extend a pattern) Science: 2.1b (planning investigations); 2.6b (analyzing data to recognize patterns) Social Studies: 2.1 (creation of sequential timelines to show historical thinking)
3.1 The student will construct sets of step-by-step instructions (algorithms), both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and	English: 3.1b (presenting instructions); 3.8 (writing structured instructions) Mathematics: 3.16 (identify/create and describe repeating and growing patterns using words, objects, pictures, numbers, and tables)

Computer Science Standard	Opportunity for Integration
c) using events.	Science: 3.1b (planning investigations with procedures); 3.1d (use patterns to draw conclusions)
4.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively a) using sequencing; b) using loops; c) using variables to store and process data; and d) performing number calculations on variables (e.g., addition, subtraction, multiplication and division).	English: 4.7a (engage in writing of sequences) Mathematics: 4.1c (students write an algorithm to round numbers); 4.4d (solving multistep problems) 4.5c (solving single step problems); 4.6b (solving multistep problems) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system); 4.1b (planning investigations) Social Studies: 4.1 (sequencing events in VA history)
5.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively, a. using sequencing; b. using loops; c. using variables to store and process data; d. performing number calculations on variables (addition, subtraction, multiplication and division); and e. using conditionals (if-statements).	Mathematics: 5.2a (determining inequalities); 5.3 (classifying prime/composite/even/odd numbers by their characteristics); 5.18 (creating and describing patterns); 5.19 (describing and using variables) Science: 5.1a (defined design problems that can be solved through the development of an object, tool, process, or a system); 5.1b (planning investigations and writing procedures) Social Studies: VS.1c (create a timeline of events in sequence)

ALPR Module 2. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- **How to Teach Sequencing Skills to Children - Speech And Language Kids (https://www.speechandlanguagekids.com/teach-sequencing-skills-children/)**
- **Sequence Facts for Kids | KidzSearch.com (https://wiki.kidzsearch.com/wiki/Sequence)**
- **What is Sequence? | Coding for Kids | Kodable - YouTube (https://www.youtube.com/watch?v=v_Pc3UnePZY)**
- **Examples of Real-Life Arithmetic Sequences – Time Flies Edu (https://timefliesedu.com/2019/12/17/examples-of-real-life-arithmetic-sequences/)**
- **Sequencing of Events - STRATEGIES (weebly.com) (https://strategiesforspecialinterventions.weebly.com/sequencing-of-events.html)**
- **How to teach kids number patterns and sequences (+5 worksheet bundles to lean on) (kidskonnect.com) (https://kidskonnect.com/articles/how-to-teach-kids-number-patterns-and-sequences/)**
- **Repeating And Growing Patterns - Bing video (https://www.bing.com/videos/search?q=growing+vs+repeating+pattern&docid=608008472586293169&mid=33A13306881B3466C34533A13306881B3466C345&view=detail&FORM=VIRE)**

Published | Preview | Assign To | Edit

ALPR Module 2 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 70 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	98
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

Due	For	Available from	Until
-	Everyone	-	-

ALPR Module 2 Assessment

This is a preview of the published version of the quiz

Started: Sep 19 at 4:50pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 70 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Question 1 14 pts

In a pattern of any sort, things must repeat, or repeatedly grow.

- True
- False

Question 2 14 pts

Often on our planners and calendars we mark an event (e.g. a class or meeting) as “recurring” with an appropriate frequency e.g. (“daily” or “biweekly”).

The concept in algorithms and programming this most closely corresponds is:

- Data storage
- Variables
- Looping
- Conditional statement

Question 3 14 pts

Below are some commands for a robot to make a ham sandwich. However, the commands are not in the proper sequence.

All ingredients are in open containers for the robot to access. Order the steps required for the robot to successfully make the sandwich.

A. Grab a slice of ham

B. Grab a slice of bread

C. Put the ham on top of the bread

D. Get a clean plate

E. Add the slice of bread to the plate

F. Add the slide of bread to the sandwich

1 [Choose]

2 [Choose]

3 [Choose]

4 [Choose]

5 [Choose]

6 [Choose]

7 [Choose]

Question 4 14 pts

Consider the Scratch program below.

What is the final value of Counter ?

- 6
- 0
- 8
- 4
- 10

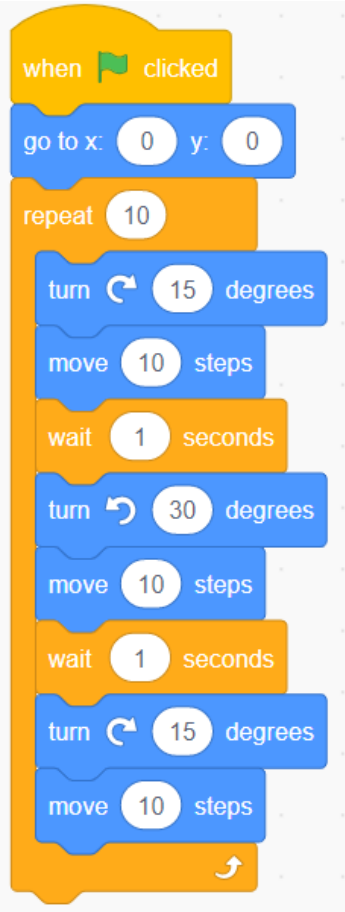
Question 5 14 pts

Sometimes the order in which the things are done makes a difference, at other times it doesn't. Identify all the scenario(s) where order matters:

- [ ] Doing roll-call to ensure that all students enrolled in the class are present.
- [ ] Following instructions of a recipe to bake a cake.
- [ ] Searching for items to buy while shopping online.
- [ ] Dialing the digits of a phone number.

Question 6 14 pts

The two programs below accomplish the same movement:



- ○ True
- ○ False

Question 7 14 pts

Select all examples of events that start a program.

- ☐ Thermostat set to “Cool - 72 degrees” detecting room temperature of 73
- ☐ Reducing pressure on the gas pedal to decelerate the car
- ☐ Placing one’s hands under an automatic air dryer
- ☐ Pressing the power button

Not saved Submit Quiz

Return to Course Materials

Congratulations, you have completed this module! **Click here to return to Course Materials (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)** .

Algorithms and Programming Module 3

This learning module will focus on **Algorithms**.

When you complete this learning module, you should be able to:

- **Construct** algorithms to include structures such as loops, variables, and conditionals.
- **Explain** different types of creative products that can be generated using a computing device (e.g., computer games, interactive stores, graphic design, programs, music, and movies).
- **Determine** an original problem and create a solution using a text or block-based program.

ALPR Module 3. Teacher Competencies and Alignment with CS SOLs

This learning module includes three teacher competencies (in bold) which are aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 3. Construct algorithms to include structures such as loops, variables, and conditionals.

ALPR Teacher Competency 4. Explain different types of creative products that can be generated using a computing device (e.g., computer games, interactive stores, graphic design, programs, music, and movies).

ALPR Teacher Competency 6. Determine an original problem and create a solution using a text or block-based program.

- CS K.1. The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing, emphasizing the beginning, middle, and end.
- CS K.2. The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end.
- CS 1.1. The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively, including
  a) sequencing (including ordinal numbers) and;
  b) simple loops (patterns and repetition).
- CS 1.2. The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively including
  a) sequencing, ordinal numbers; and
  b) simple loops (patterns and repetition).
- CS 2.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and,
  c) identifying events.
- CS 2.2. The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns, such as repeating patterns or growing patterns); and
  c) identifying events.
- CS 3.1. The student will construct sets of step-by-step instructions (algorithms), both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and
  c) using events.
- CS 3.2. The student will construct programs to accomplish tasks as a means of creative expression using a block or text based programming language, both independently and collaboratively
  a) using sequencing;
  b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and
  c) identifying events.
- CS 3.4. The student will create a plan as part of the iterative design process, independently and/or collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).

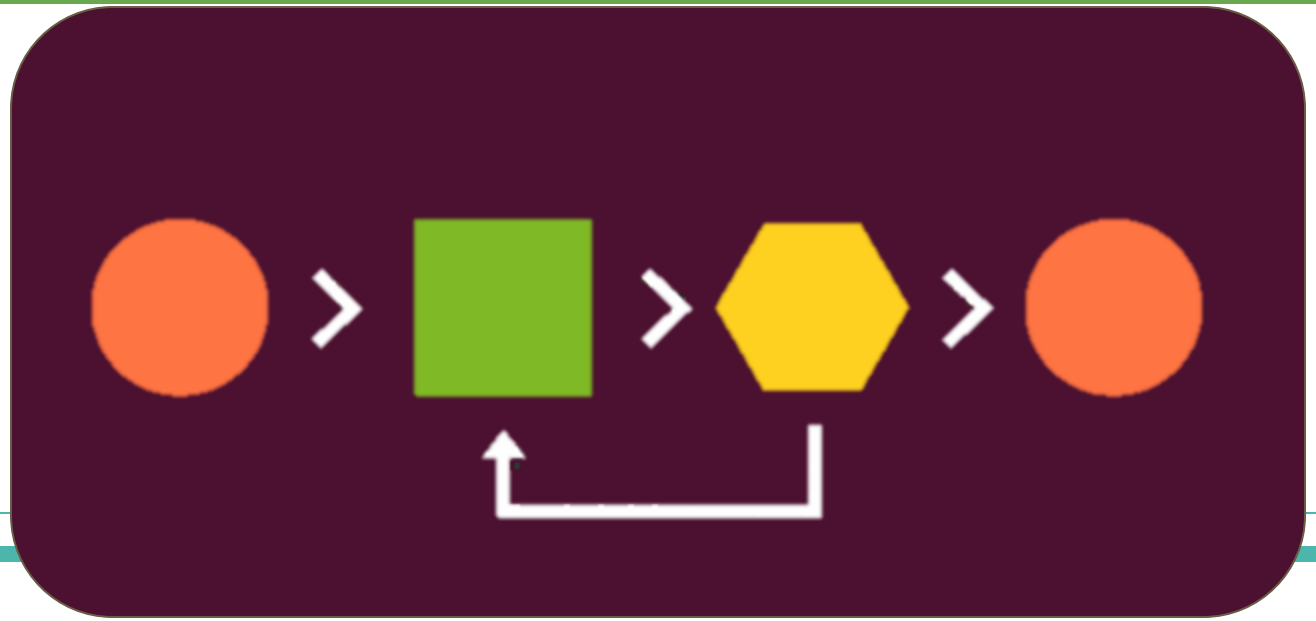
- CS 4.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively
  a) using sequencing;
  b) using loops;
  c) using variables to store and process data; and
  d) performing number calculations on variables (e.g., addition, subtraction, multiplication and division).
- CS 4.2. The student will construct programs to accomplish a task as a means of creative expression using a block or text based programming language, both independently and collaboratively
  a) using sequencing;
  b) using loops;
  c) using variables; and
  d) performing number calculations (e.g., addition, subtraction, multiplication and division) on variables.
- CS 4.4. The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).
- CS 5.1. The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively,
  a) using sequencing;
  b) using loops;
  c) using variables to store and process data;
  d) performing number calculations on variables (addition, subtraction, multiplication and division); and
  e) using conditionals (if-statements).
- CS 5.2. The student will construct programs to accomplish a task as a means of creative expression using a block or text based programming language, both independently and collaboratively
  a) using sequencing;
  b) busing loops;
  c) using variables;
  d) using mathematical operations (addition, subtraction, multiplication and division) variable to manipulate a variable; and
  e) using conditionals (if-statements).
- CS 5.4. The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).

Algorithms

View the lesson video first, then view the activity video. A PDF of the slides and a transcript of the videos is also included for accessibility.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

3.2 ADVANCED ALGORITHMS

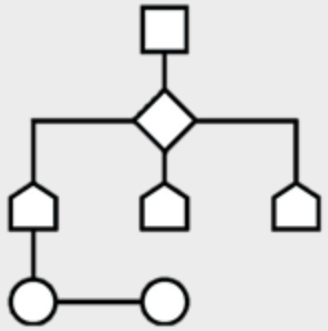


Slide 1
Hello and welcome to the ARCS Microcredential of Algorithms and Programming. We're going to be getting into advanced algorithms, which you have already been working with. We will learn a little something that we can do to practice with algorithms.

ADVANCED ALGORITHMS

Algorithms can have BRANCHES and loops,
which can make them **MORE complicated**.

Identifying the **beginning**, **middle**, and **end** of an algorithm
not only helps with **reading** someone else's code,
but with **designing** and **planning** of your **own**!



Slide 2
Here we are with advanced algorithms. Algorithms can have branches and loops, which can make them more complicated. Identifying the beginning, middle, and end of an algorithm or sequence not only helps with reading someone else's code, but with designing and planning of your own.

Identifying Parts of a SEQUENCE/ALGORITHM

Every time you build a project in Scratch
you indirectly identify the *flow* of sequences.

In section 2, we did an activity where we **planned** the **steps** of our sequence before building it in Scratch.

In this section, we will look at **already built algorithms** and identify the **order** of the **steps** along with **what it does.**

Slide 3

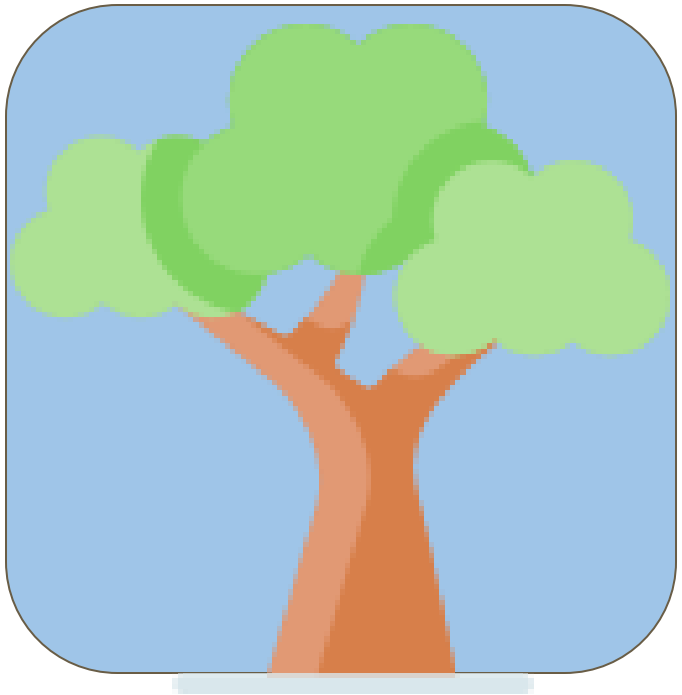
We will now learn about identifying parts of a sequence or algorithm. Every time you build a project in Scratch, you indirectly identified the flow of sequences. In Section two, we did an activity where we planned the steps of our sequence before putting it in Scratch. In this section, we will look at already-built algorithms and identify the order of steps along with what it does.

Follow the Program!

💻 There are **three** prebuilt programs.

🦶 Write, **in order** (**style is up to you**), the steps each program takes.

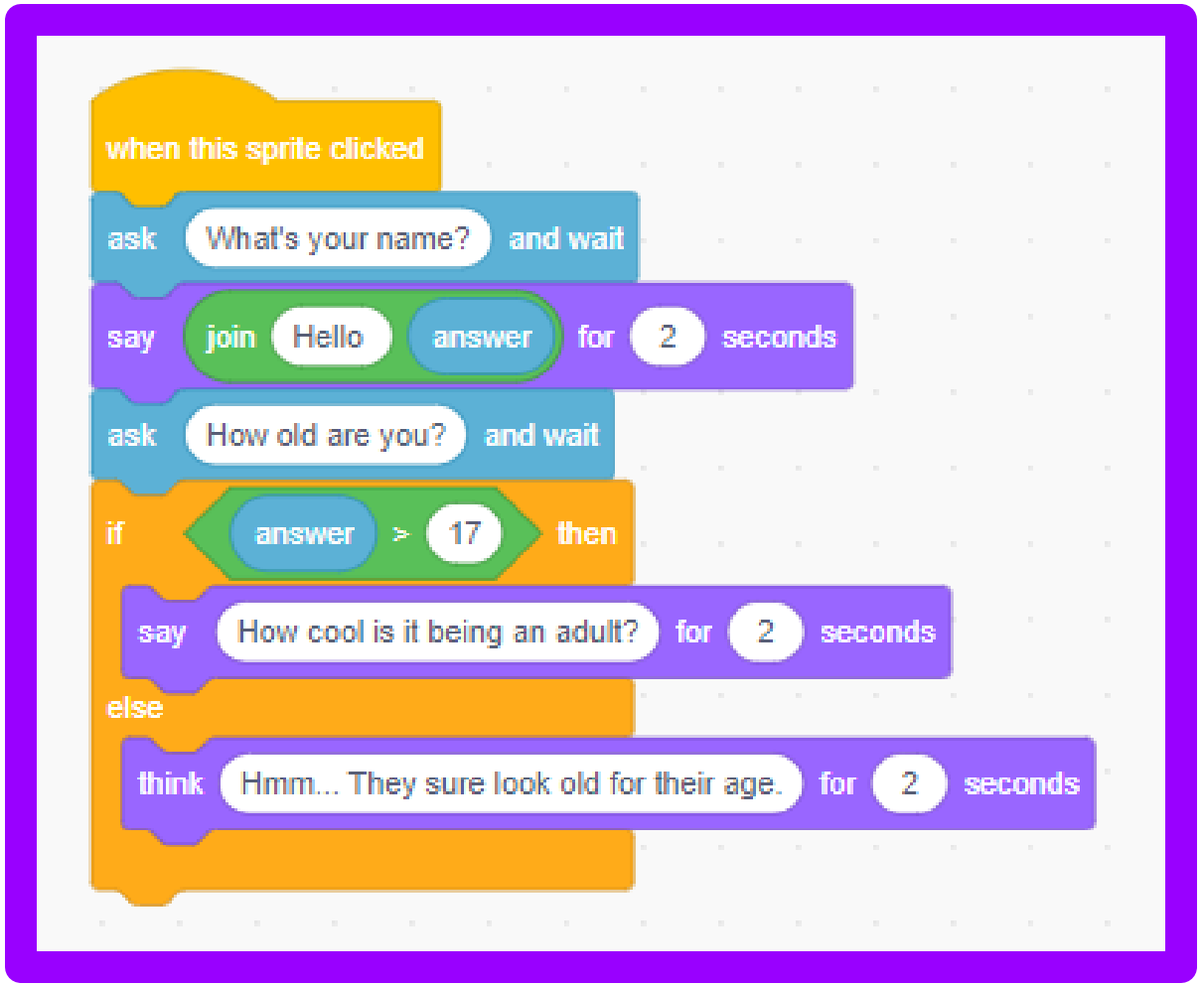
🌳 For the decision statements, ***try making the decision tree***.



Slide 4

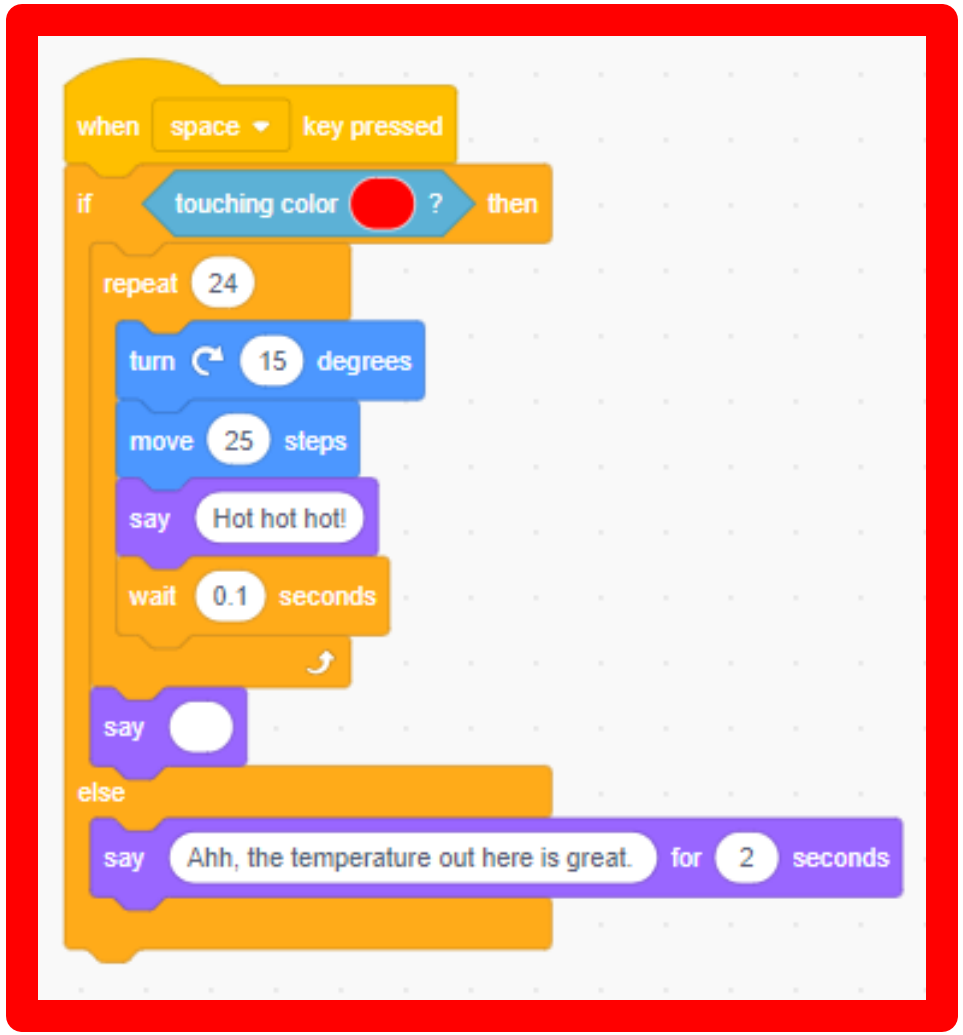
Follow the program. There are three pre-built programs; the style is up to you for the steps each program takes in the decision statement. Try making the decision tree. We're going to write this out on a piece of paper. Or if you want, you can use Microsoft Word, or maybe even try to build it in PowerPoint. It's whatever you want to do; again, the style is up to you. In the next video, we're going to go over the correct ways that these programs flow. I'll see you there.

THE ADULT BLOCK



Slide 5
This is a part of the video where you can pause and work ahead if you'd like. Or we can work through it together, or you can simply watch. Let's go ahead and get started. First we have what I have called the "adult block" when this Sprite is clicked. Let's work through this program. It all starts off with when this Sprite is clicked. Next we ask "what is your name"? After waiting, we say "Hello [answer]". We wait for two seconds and ask, "How old are you?" and wait again. Tf answer is greater than 17, we say "how cool is it being an adult?"; otherwise, in the "else", we say "they sure look old for their age". This probably looks very familiar as it was in the last project that we did. If you wrote out the decision tree for this, good job. That concludes the adult block.

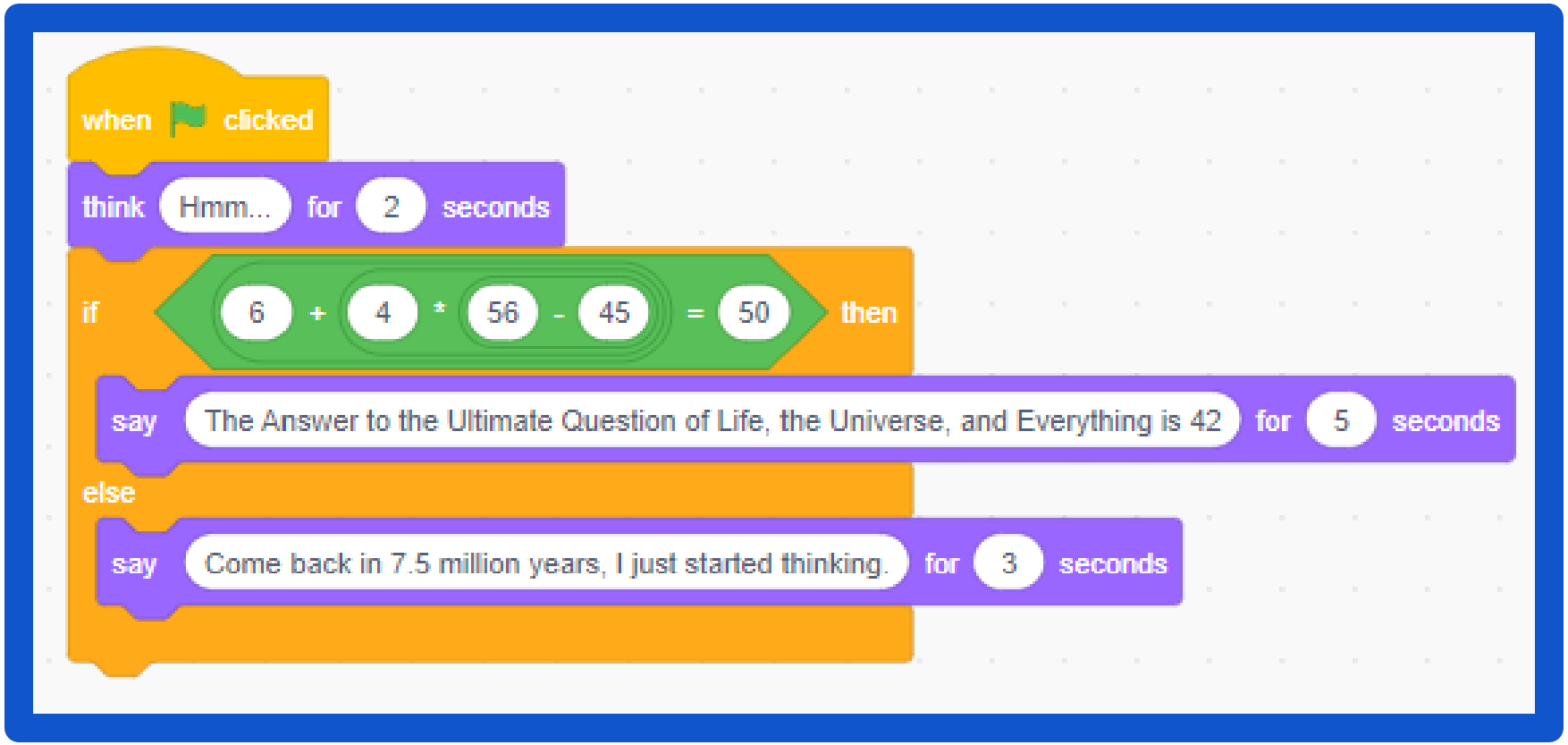
THE TEMPERATURE BLOCK



Slide 6

Let's look at the next one, which I have called the "temperature block". When the space key is pressed, if the Sprite is touching the color red, then repeat this 24 times: turn 15 degrees, move 25 steps, say "hot, hot, hot!", and wait 0.1 seconds. In reality, this turns into 24 seconds of turning, moving, and saying "hot hot, hot". Put this in the repeat block and have it say nothing after - clears out the say block. If not touching the color red ("else"), we say, "Ah, the temperature out here is great". And that concludes the temperature block.

THE ANSWER TO LIFE BLOCK



Slide 7

Lastly, we have the "answer to life" block. When the green flag is clicked, think for two seconds. I made this one intentionally complicated - I'm very sorry. The way that these blocks work is the innermost block goes first, then moves its way out. First, we do 56 minus 45, which is going to be 11, which we then multiply by four to make 44, and then add six to equal 50. We then say "if 50 equals 50", then the returned response will be "Ah, the answer to the ultimate question of life, the universe and everything is 42". In not ("else") we say "come back in 7.5 million years. I just started thinking". Those are the three blocks. I hope you were able to write them out without any trouble! I'll see you in the next video.

ALPR Module 3. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 3. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of these CS SOL competencies across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Algorithms.pdf
(https://canvas.odu.edu/courses/185316/files/44845397/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845397/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 3. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
K.1 The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing, emphasizing the beginning, middle, and end.	English: K.1 (oral communication); K.8 (sequence stories using beginning, middle, and end) Mathematics: K.2 (sequence sets); K.13 (patterns) Science: K.1 (across all standards); K.3 (sequencing objects); K.9 (patterns in nature); K.10 (change) Social Studies: K.1 (sequence events); K.1 (collaborating and participating in classroom activities)
K.2 The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end.	Science: K.3 (sequencing objects) Social Studies: K.1 (collaborating and participating in classroom activities)
1.1 The student will construct sets of step-by-step instructions (algorithms) either independently or collaboratively, including a) sequencing (including ordinal numbers) and; b) simple loops (patterns and repetition).	English: 1.1i (giving simple directions); 1.2b (tell stories in sequential order) Mathematics: 1.2c (placing numbers in order); 1.3 (placing numbers in order); 1.14 (growing and repeating patterns) Science: 1.1b (planning investigations)
1.2 The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively including a) sequencing, ordinal numbers; and b) simple loops (patterns and repetition).	English: 1.1i (giving simple directions); 1.2b (tell stories in sequential order) Mathematics: 1.2c (placing numbers in order); 1.3 (placing numbers in order, ordinal position); 1.14 (repeating patterns) Science: 1.1b (planning investigations)
2.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and, c) identifying events.	English: 2.1k (give and follow multi-step directions); 2.2a (use the story structure of beginning, middle, and end to tell a story of an experience); 2.10e (organize writing to include a beginning, middle, and end) Mathematics: 2.2 (determining patterns); 2.16 (describe the core of a repeating pattern, extend a

Computer Science Standard	Opportunity for Integration
	pattern) Science: 2.1b (planning investigations); 2.6b (analyzing data to recognize patterns) Social Studies: 2.1 (creation of sequential timelines to show historical thinking)
2.2 The student will construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns, such as repeating patterns or growing patterns); and c) identifying events.	English: 2.1k (give and follow multi-step directions); 2.2a (use the story structure of beginning, middle, and end to tell a story of an experience); 2.10e (organize writing to include a beginning, middle, and end) Mathematics: 2.2 (determining patterns); 2.16 (describe the core of a repeating pattern, extend a pattern) Science: 2.1b (planning investigations); 2.6b (analyzing data to recognize patterns)
3.1 The student will construct sets of step-by-step instructions (algorithms), both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and c) using events.	English: 3.1b (presenting instructions); 3.8 (writing structured instructions) Mathematics: 3.16 (identify/create and describe repeating and growing patterns using words, objects, pictures, numbers, and tables) Science: 3.1b (planning investigations with procedures); 3.1d (use patterns to draw conclusions)
3.2 The student will construct programs to accomplish tasks as a means of creative expression using a block or text based programming language, both independently and collaboratively a) using sequencing; b) using loops (a wide variety of patterns such as repeating patterns or growing patterns); and c) identifying events.	English: 3.1b (presenting instructions); 3.8 (writing structured instructions) Mathematics: 3.16 (identify/create and describe repeating and growing patterns using words, objects, pictures, numbers, and tables) Science: 3.1b (planning investigations with procedures); 3.1d (use patterns to draw conclusions) Social Studies: 3.1 (organizing information into timelines)
3.4 The student will create a plan as part of the iterative design process, independently and/or collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 3.8c (use a variety of prewriting strategies to plan and organize writing) Science: 3.1a (define a simple problem); 3.1b (use tools and/or materials to design and/or build a device to solve a specific problem); 3.1c (analyze data from tests of an object or tool to determine if it works as intended); 3.1f (communicate design ideas and/or solutions with others)
4.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively a) using sequencing;	English: 4.7a (engage in writing of sequences) Mathematics: 4.1c (students write an algorithm to round numbers); 4.4d (solving multistep problems)

Computer Science Standard	Opportunity for Integration
b) using loops; c) using variables to store and process data; and d) performing number calculations on variables (e.g., addition, subtraction, multiplication and division).	4.5c (solving single step problems); 4.6b (solving multistep problems) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system); 4.1b (planning investigations) Social Studies: 4.1 (sequencing events in VA history)
4.2 The student will construct programs to accomplish a task as a means of creative expression using a block or text based programming language, both independently and collaboratively a) using sequencing; b) using loops; c) using variables; and d) performing number calculations (e.g., addition, subtraction, multiplication and division) on variables.	English: 4.7a (engage in writing of sequences) Mathematics: 4.1c (students write an algorithm to round numbers); 4.4d (solving multistep problems); 4.5c (solving single step problems); 4.6b (solving multistep problems); 4.15 (patterns) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system); 4.1b (planning investigations – writing procedures)
4.4 The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 4.7d (using prewriting strategies) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system); 4.1b (use tools and/or materials to design and/or build a device that solves a specific problem); 4.1c (analyze data from tests of an object or tool to determine if it works as intended); 4.1e (identify limitations of a model); 4.1f (communicate design ideas or solutions to others)
5.1 The student will construct sets of step-by-step instructions (algorithms) both independently and collaboratively, a. using sequencing; b. using loops; c. using variables to store and process data; d. performing number calculations on variables (addition, subtraction, multiplication and division); and e. using conditionals (if-statements).	Mathematics: 5.2a (determining inequalities); 5.3 (classifying prime/composite/even/odd numbers by their characteristics); 5.18 (creating and describing patterns); 5.19 (describing and using variables) Science: 5.1a (defined design problems that can be solved through the development of an object, tool, process, or a system); 5.1b (planning investigations and writing procedures) Social Studies: VS.1c (create a timeline of events in sequence)
5.2 The student will construct programs to accomplish a task as a means of creative expression using a block or text based programming language, both independently and collaboratively a. using sequencing; b. using loops; c. using variables; d. using mathematical operations (addition, subtraction, multiplication and division) variable to manipulate a variable; and	Mathematics: 5.2a (determining inequalities); 5.3 (classifying prime/composite/even/odd numbers by their characteristics); 5.18 (creating and describing patterns); 5.19 (describing and using variables) Science: 5.1a (defined design problems that can be solved through the development of an object, tool, process, or a system); 5.1b (planning investigations and writing procedures)

Computer Science Standard	Opportunity for Integration
e. using conditionals (if-statements).	Social Studies: VS.1c (create a timeline of events in sequence)
5.4 The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 5.1 (preparing prewriting tools); 5.7c (using a variety of prewriting tools) Science: 5.1a (defined design problems that can be solved through the development of an object, tool, process, or a system); 5.1b (planning investigations and writing procedures); 5.1c (use data to evaluate and refine design solutions); 5.1d (generate and compare multiple solutions to problems based on whether they meet criteria and constraints); 5.1e (identify limitations of models); 5.1f (communicate design ideas or solutions to others)

ALPR Module 3. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- Algorithmic Thinking: Loops and Conditionals (cmu.edu) (https://www.cs.cmu.edu/~15110-n15/lectures/unit03-Algorithm-1.pdf)
- Conditionals and How They Are Used in Algorithms | by Sean Guthrie | Medium (https://medium.com/@seanguthrie/conditionals-and-how-they-are-used-in-algorithms-9c98d35b57c)
- What is a Variable in Programming? - Definition from Techopedia (https://www.techopedia.com/definition/25647/variable-programming)
- Introduction to Algorithms - GeeksforGeeks (https://www.geeksforgeeks.org/introduction-to-algorithms/)
- 10 Everyday Objects That Can Be Programmed To Run Code (fastcompany.com) (https://www.fastcompany.com/3016427/10-everyday-objects-that-can-be-programmed-to-run-code)
- List of computer simulation software - Wikipedia (https://en.wikipedia.org/wiki/List_of_computer_simulation_software)
- Top 10 Uses of Computers in our daily life | KLIENT SOLUTECH (https://www.klientsolutech.com/top-10-uses-of-computers-in-our-daily-life/)
- Scratch - Imagine, Program, Share (mit.edu) (https://scratch.mit.edu/projects/editor/?tutorial=getStarted)
- Block-based coding (codejig.com) (https://www.codejig.com/en/block-based-coding/)
- 5. if and for Statements | Programming for Beginners (programming4beginners.com) (https://www.programming4beginners.com/tutorial/chapter05/if-and-for-statements)
- How to Solve Coding Problems with a Simple Four Step Method (freecodecamp.org) (https://www.freecodecamp.org/news/how-to-solve-coding-problems/)

Published | Preview | Assign To | Edit

ALPR Module 3 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 80 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	99
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

ALPR Module 3 Assessment

ⓘ This is a preview of the published version of the quiz

Started: Sep 19 at 5:02pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 80 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Question 1 9 pts

Consider the following three sequences:

i. A, B, A, B, A, B
ii. ★■, ★★■, ★★★■
iii. 12, 23, 34, 45, 56, 67

Which of the following statements is true?

○ (i) and (ii) have a repeating pattern

○ (ii) and (iii) have a growing pattern

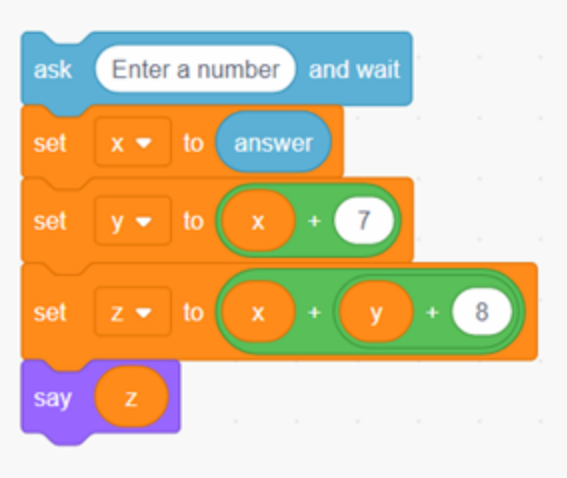
○ (i) and (iii) have a growing pattern

○ (i) has a repeating pattern but (iii) has no discernible pattern

○ None of the above

Question 2 9 pts

The following is a simple program with four steps to read two numbers and display the value of an expression calculated using the values of the two numbers.



The only variables in this program are:

- x
- x and y
- y
- x, y, and z
- None of the above

Question 3 9 pts

Which of the following are true in programming? Select all that apply.

- [ ] Variables are used for storing and manipulating values used in calculation.
- [ ] Names of variables can change when the program is executed.
- [ ] Values of variables can change when the program is executed.
- [ ] Variables with different names can't have the same value.
- [ ] If the value of a variable doesn't change, it is not a variable.

Question 4 9 pts

For the next two items, consider that Alice is writing a computer program to compare the speeds of two cars that have raced one another.

Review Alice's Scratch problem below and mark all statements that are true about her program:

- [ ] It does not allow speedA and speedB be the same value
- [ ] It always displays the larger of the speedA and speedB if unequal
- [ ] It never displays 0 because both speedA and speedB are positive numbers
- [ ] It always displays 0 because fastestCar is set to 0.
- [ ] It uses conditional statements but no looping statements

☐
It always displays a value that equals the value of speedA or speedB

Question 5 9 pts

Which of the following is most likely to be devoid of any pattern:

○
Endings of lines in a nursery rhyme

○
Outcomes of rolls of a dice

○
Temperatures measured on each day of the year

○
Dates on which we observe full moons

Question 6 9 pts

Ms. Turing has 20 students in her English class. She writes the following computer program that will calculate and print the average score of the students on their first quiz.

Set *Total* to 0.

Repeat 20 times:

- **Read the next student's score.**
- **Update *Total* by adding this score to *Total*.**

Set *Average* to *Total*/20.

Print *Average*.

This program uses:

○
Variable(s) and looping structure(s), but not conditional statement(s)

○
Looping structure(s) and conditional statement(s), but no variable(s)

○
Variable(s), looping structure(s), and conditional statement(s)

Question 7 9 pts

The Scratch program below compares you and your friend's dice roll.

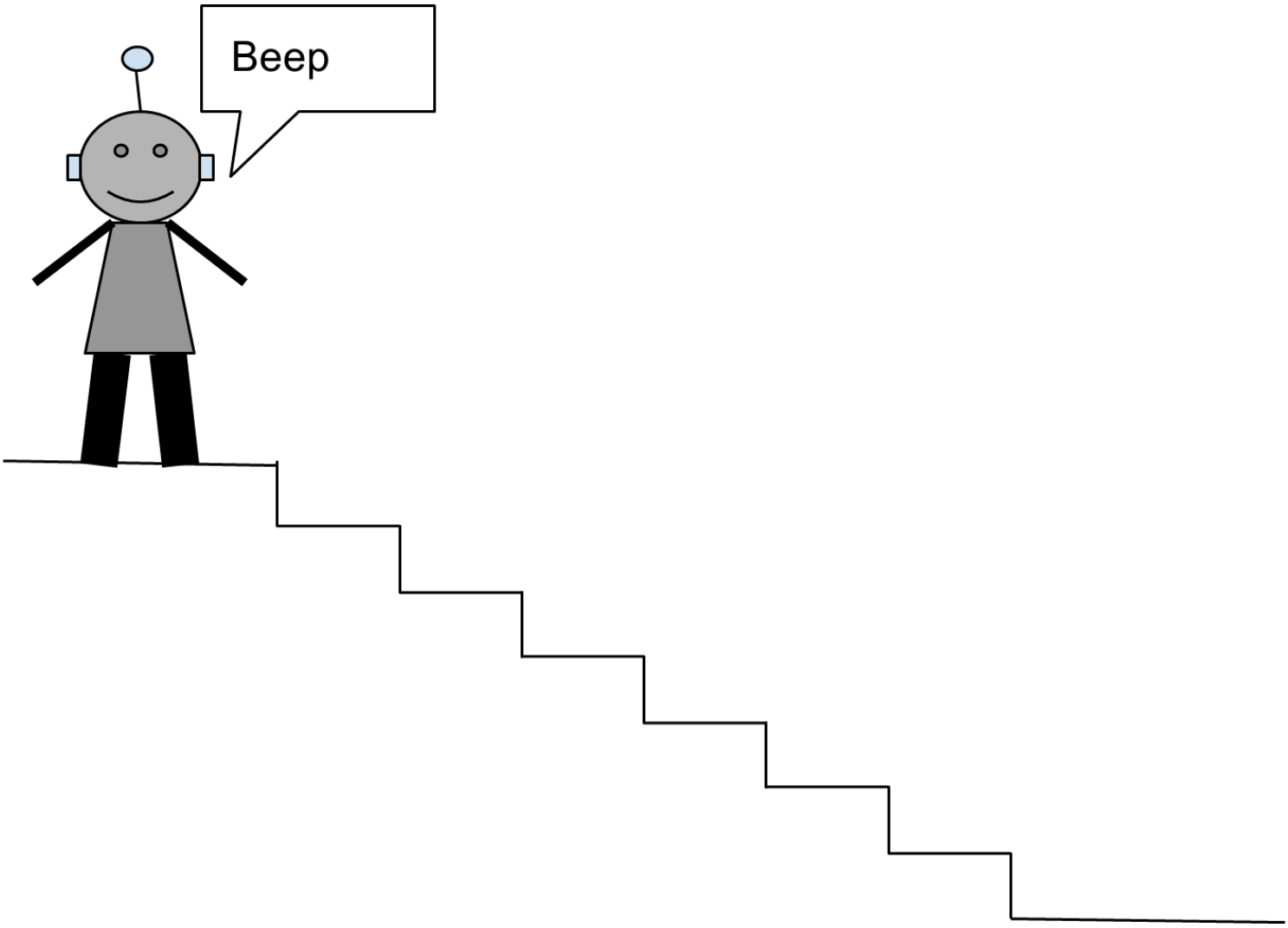
Which condition should replace the empty area of the "if" statement?

- Their Roll > Your Roll
- Your Roll > Their Roll
- Your Roll = Their Roll
- Your Roll < Their Roll

Question 8 9 pts

Please reference the image of the robot below to answer this and the next three questions.

This robot needs to get down the flight of stairs. A programmer writes the code for the robot to go down one step. The program will run the code repeatedly until the robot reaches the bottom.



This repetition of code is called a(n) _____.

- Variable
- Loop
- Event
- Operation

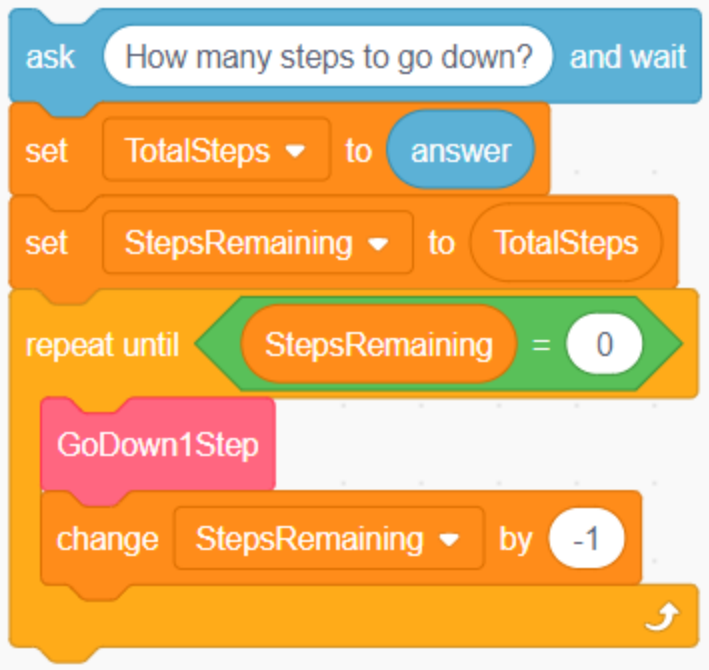
Question 9 9 pts

How many times will it appear?

- 0
- 1
- 6
- 7

Question 10 9 pts

Consider the following Scratch program below helping the robot get down the stairs.



Which variable intially declares the number of steps to get down the stairs?

- *TotalSteps*
- *StepsDown*

StepsRemaining

Question 11 9 pts

Match the conditional on *StepsRemaining* to the proper decision:

If StepsRemaining is greater than 0, then [Choose]

If StepsRemaining is equal to 0, then [Choose]

If StepsRemaining is less than 0, then [Choose]

Not saved Submit Quiz

Algorithms and Programming Module 4

This learning module will focus on **Classification**.

When you complete this learning module, you should be able to:

- **Classify** and **arrange** items based on their attributes, actions, and patterns.

ALPR Module 4. Teacher Competencies and Alignment with CS SOLs

This learning module includes one teacher competency (in bold), which is aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 5. Classify and arrange items based on their attributes, actions, and patterns.

- CS K.4. The student will categorize a group of items based on one attribute or the action of each item, with or without a computing device.
- CS 1.5. The student will categorize a group of items based on one or two attributes or the actions of each item, with or without a computing device.
- CS 2.5. The student will compare and contrast a group of items based on the attributes or actions of each item, with or without a computing device.
- CS 3.5. The student will compare and contrast a group of items based on attributes or actions classified into at least two sets and two subsets.
- CS 4.5. The student will classify and arrange a group of items based on the attributes or actions.

Classification - Lesson materials

Why is Classification an Important Concept in Algorithms and Programming?

In many cases, the steps of an algorithm can **change** based on **different conditions**. These differences in condition are determined through **classification**. This allows an algorithm to **adjust** steps and results.

Consider this example: we all have our daily algorithm for getting dressed in the morning. We all determine what is appropriate to wear by **classifying** the weather. What's the **color** of the sky? Is it blue or grey? Does the weather feel **cold**, **hot**, or **warm**? Is it **bright** outside or **cloudy**? Can you hear the **sound** of rain or thunder? Is the sidewalk **covered** in snow, and if so, **how much**? All of these questions can be used to classify the weather, which is used to determine what clothing one will be wearing.

If the sky is a bright blue color, the air feels warm, yet not hot, and the birds can be heard flying and chirping, then our algorithm would likely take that classification and determine that a light sweater would be perfect. If raindrops can be felt dropping and the weather is chilly, a jacket would work.

How about deciding if somebody can run for **President** of the United States. We first have to classify them by their **age**: are they 35 or older, or under 35? If they are under 35, the algorithm determines they cannot be president. After that, we use the algorithm by classifying whether the person is **born in the USA** or not. If they are, they can be run for presidency; otherwise, they cannot. The process of figuring out somebody's eligibility to be president is an **algorithm** itself!

Without classification, algorithms would not be able to adjust and change based on different conditions, making the ***Classification*** lesson a very critical part of this Microcredential.

GROUPING

By Attributes

Slide 1

Hello, everyone, and welcome to the ARCS Microcredential for Algorithms and Programming: Grouping by Attributes. We've already seen a good example of this from our Introduction to Scratch, as all of the code blocks are grouped based on how they affect the program. The perfect example of that is motion blocks make things move, looks blocks effect the look of the program, control blocks control the program. Let's go ahead and move on.

An ATTRIBUTE* is a QUALITY or CHARACTERISTIC.

It can be collected as DATA!

* Also known as a PROPERTY

Slide 2

What is an attribute? Well, an attribute is a quality or characteristic, and it can be collected as data. Attributes can also be called properties.

Common Attributes of Everyday Objects:

Color

Smell

Shape

Author

Size

Date

Birth, expiration, etc.

Weight

Origin

Slide 3

Common attributes of everyday objects include color, smell, shape, author, size, date, weight, and origin. These are only a few of the common ones.

Let's COMPARE attributes...

Power: Gas Engine

# of wheels: 4

# of doors: 4

Weight: 2500 pounds

Think of **other** attributes you could **compare!**

Power: Human

# of wheels: 2

# of doors: 0

Weight: 80 pounds

Slide 4

Let's compare some attributes. We've got a car. It's powered by a gas engine. This particular type of car has four wheels and has four doors. The weight is about 2,500 pounds. On the other hand, there's a bicycle, which is another type of vehicle. However, it's powered by a human, has two wheels, 0 doors, and weighs about 80 pounds. They're entirely different objects, but have similar, yet comparable attributes.

GROUPING OBJECTS BY THEIR ATTRIBUTES SERVES COUNTLESS PURPOSES IN THE REAL WORLD...

Grouping types of **groceries** into their own **aisles** makes shopping much easier and faster.

Grouping **clothing** by the washing water **temperature** and **color** prevents colors from mixing.

Grouping the same type of **assignment** in a **gradebook** helps with solving a weighted average.

Slide 5

Grouping objects by their attributes serves countless purposes in the real world. For example, grouping types of groceries into their own aisles make shopping much easier and faster. Imagine trying to look for one bag of chips and it's on one side of the store, then you want a different kind of chip, and it turns out it's on the other side of the store! Instead of that, they put all the chips in one aisle. Grouping clothing by the washing water temperature and color prevents colors for mixing. Finally, grouping the same type of assignment in a grade book helps with solving a weighted average.

Say we wanted to group this produce.

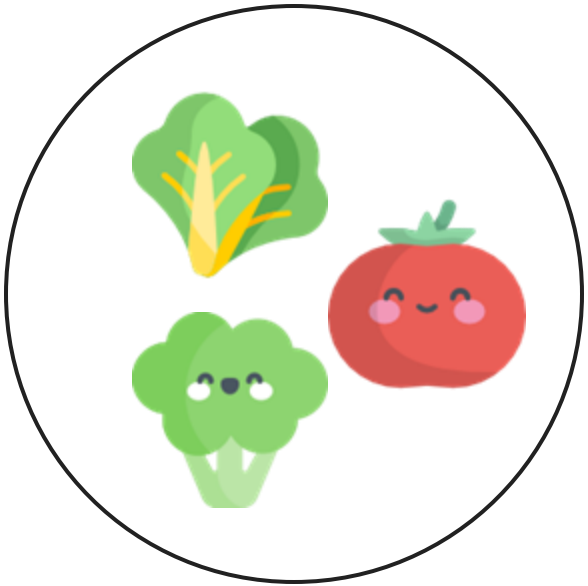





Slide 6
We want to group this produce. We have some cherries, lettuce, tomatoes, broccoli, lime, and apple.

We can group them into FRUITS and VEGETABLES*

Fruit



Vegetable

and form TWO groups of THREE types of produce each.

*Tomatoes are vegetables for this example

Slide 7

We can group them into fruits and vegetables. We have cherries, apples and limes, and then we have lettuce, tomato, broccoli. There are two groups of three types of produce each. Tomatoes are vegetables for this example, but they're actually fruits according to biology. We're not here for that.

We could group them by NAME...

Cherry

Lettuce

Tomato

Broccoli

Lime

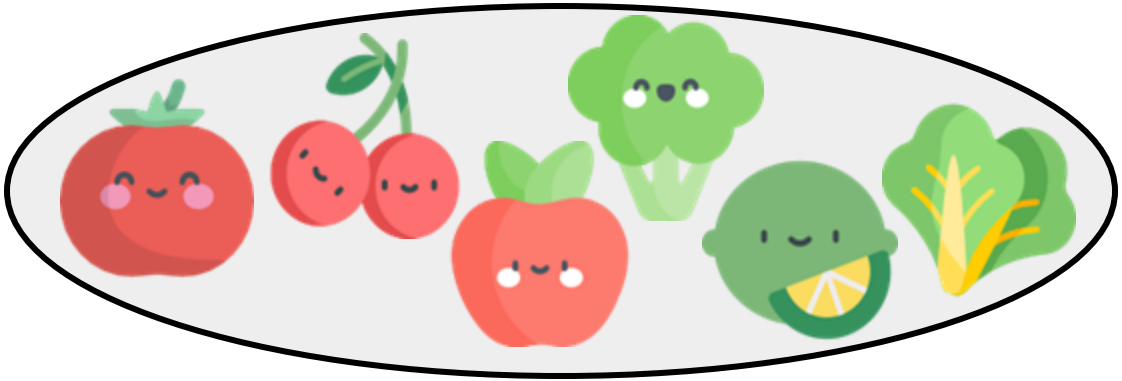
Apple

... but we have all **groups of one**, which isn't very useful.

Slide 8

We could group them by name: cherry, lettuce, tomato, broccoli, lime, and apple. But we have all groups of one, which isn't very useful.

We **could** group them by PERISHABILITY...



Perishable

Non-Perishable

... but all the produce falls into A SINGLE GROUP, which doesn't help with separating.

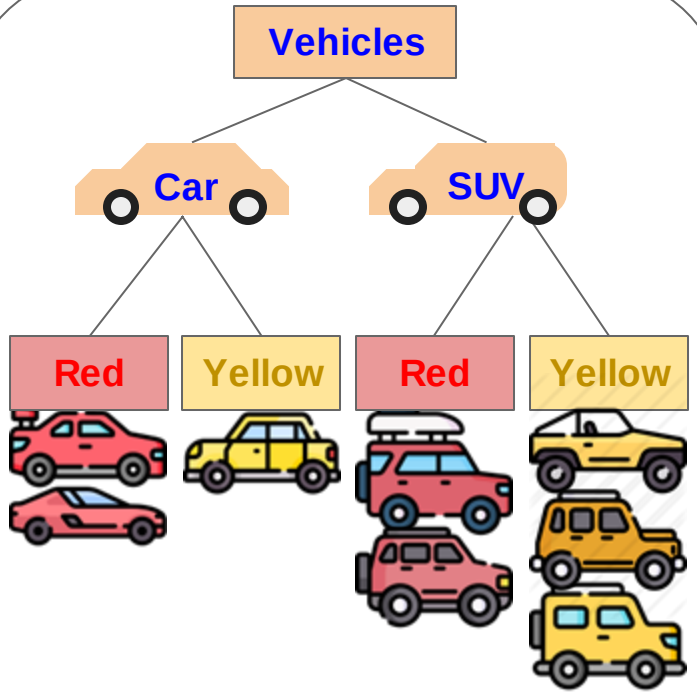
Slide 9
We could group them by perishability. Yes, they do go bad. But all the produce falls into a single group, which doesn't help with separating.

A SET OF **OBJECTS** CAN BE **GROUPED** BY

MULTIPLE ATTRIBUTES,

BUT THE **GROUPING** **MUST** BE DONE

ONE ATTRIBUTE AT A TIME!



Here, vehicles are grouped

FIRST by **TYPE**

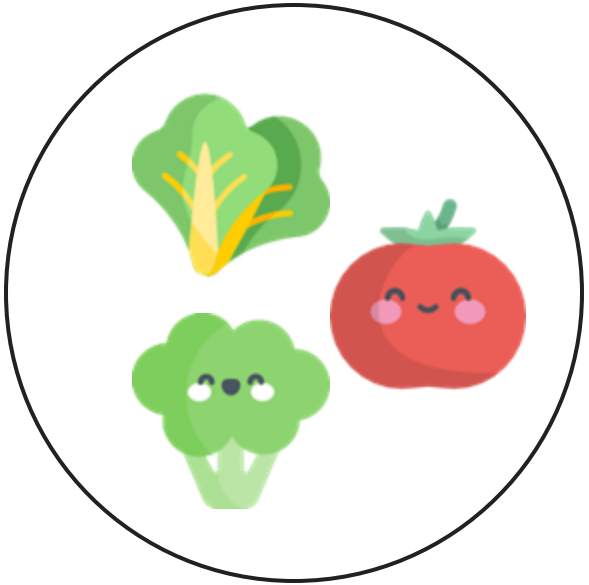
THEN by **COLOR**.

Slide 10

A set of objects can be grouped by multiple attributes, but the grouping must be done one attribute at a time. For example, we have vehicles, then we have cars and SUVs, and then we have red and yellow. So here vehicles are grouped first by type. First, the cars are put in one section and the SUVs are put in another section. Then they're grouped by color. We put our red cars together, our yellow cars together, and then our red SUVs together and our yellow SUVs together.

Looking at the grouping of FRUITS and VEGETABLES...

Fruit

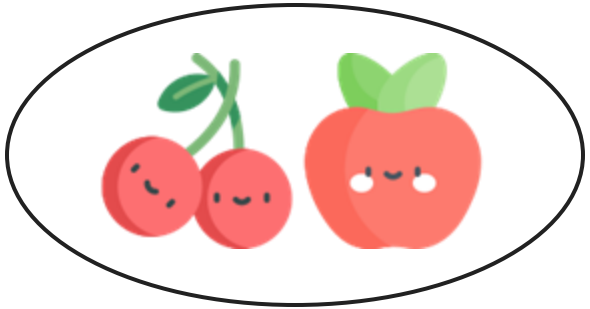


Vegetable

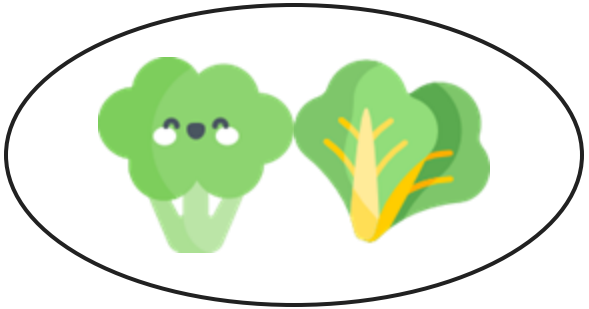
Slide 11

Looking at the grouping of fruits and vegetables, we can further group them by tanginess: non-tangy fruits and non-tangy vegetables, and then tangy fruits and tangy vegetables.

We can FURTHER group them by TANGINESS...

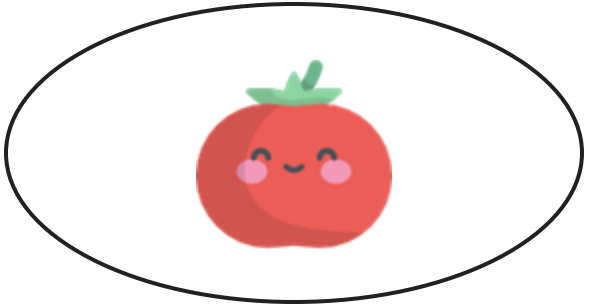


Non-Tangy Fruit



Non-Tangy Vegetable

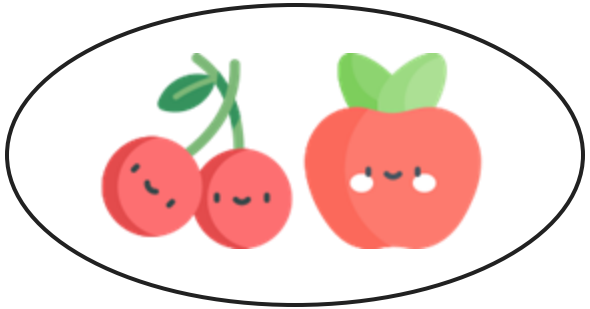
Tangy Fruit



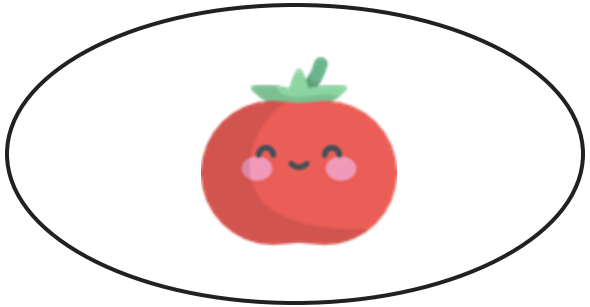
Tangy Vegetable

Slide 11
Looking at the grouping of fruits and vegetables, we can further group them by tanginess: non-tangy fruits and non-tangy vegetables, and then tangy fruits and tangy vegetables.

...OR FURTHER GROUP THEM BY COLOR INSTEAD...

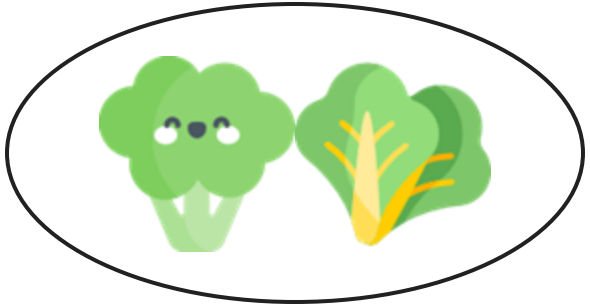


Red Fruit



Red Vegetable

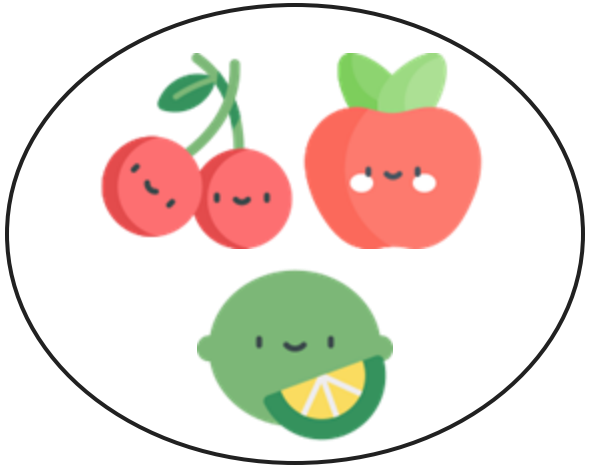
Green Fruit



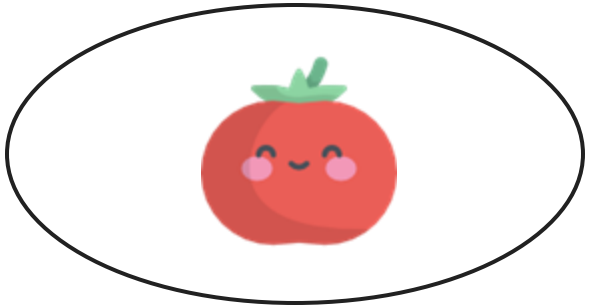
Green Vegetable

Slide 12
Or, we can further group them by color instead: red fruits, red vegetables, green fruits, green vegetables.

...OR EVEN BY SHAPE!

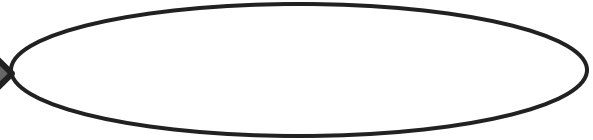


Round Fruit

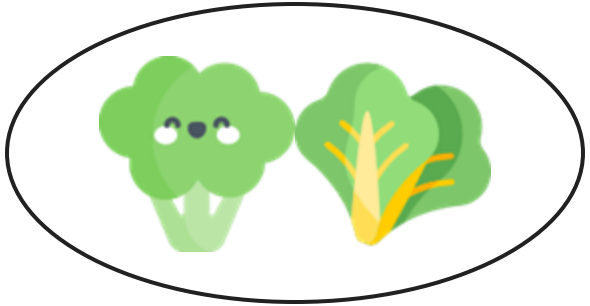


Round Vegetable

NOTICE:
NO produce fits into this category!



Non-Round Fruit



Non-Round Vegetable

Slide 13
Or, we can even do this by shape: round fruits, round vegetables, non-round fruits (notice: no produce fits in this category), and non-round vegetables.

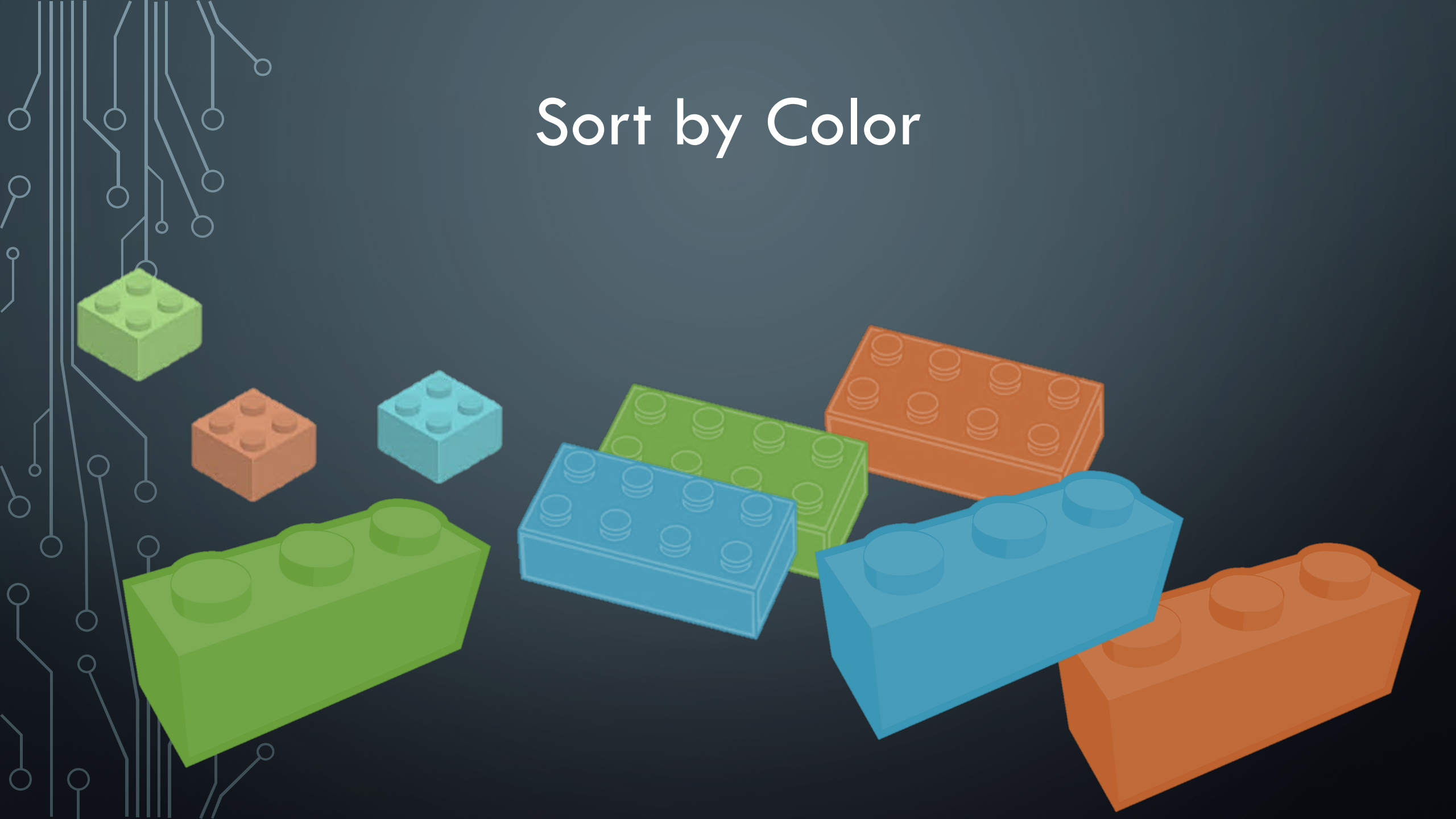
THESES GROUPINGS WERE DONE BY SPLITTING A GROUP INTO TWO GROUPS, BUT THIS CAN BE DONE WITH AS MANY GROUPS AS NEEDED TO PROPERLY CATEGORIZE OBJECTS!

Next time you go shopping, try *collecting data* from groceries in each aisle!

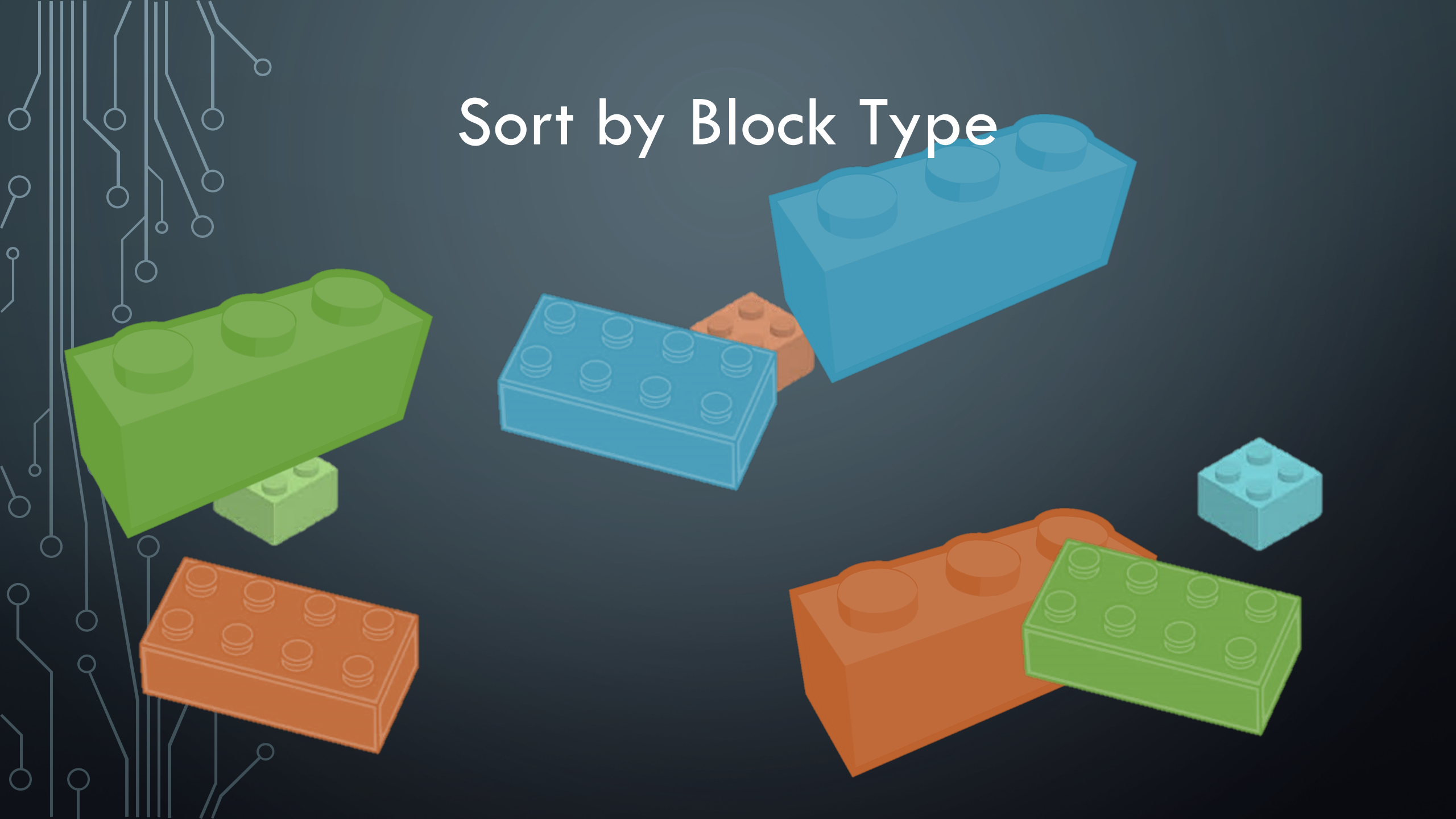
Slide 14
These groups were done by splitting a group into two groups. But this can be done with as many groups as needed to properly categorize objects. In the future, when you categorize objects, it makes them much easier to find.
We can group by attributes just like scratch does with the code blocks. Now, we have a project for categorizing and grouping objects.

Classification - Activity materials

Sort by Color

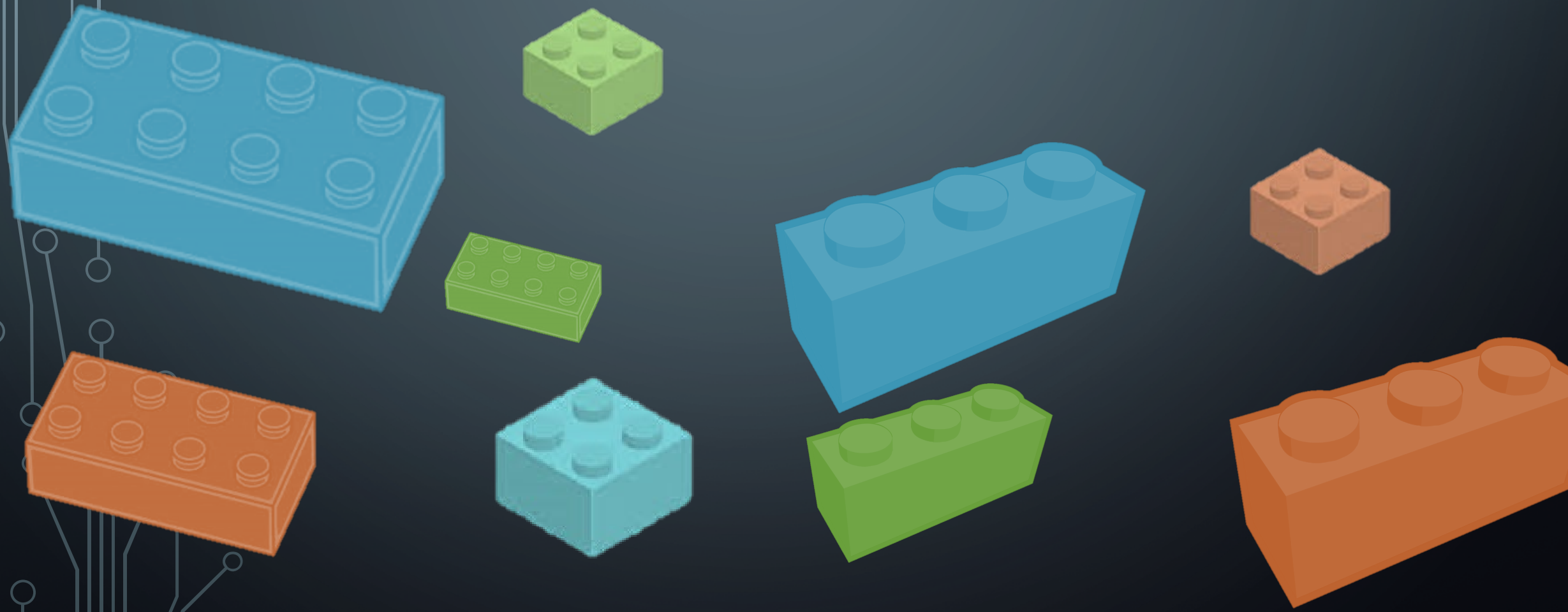


Sort by Block Type



- For the next slide sort them in a linear order
  - Left -> right
- Sort them first by size
  - Largest -> smallest
- Then sort them by color
  - Blue -> Green -> Orange

1. Sort by size
2. Sort by color



3.5 CLASSIFY AND ARRANGE ITEMS BASED ON THEIR ATTRIBUTES, ACTIONS, AND PATTERNS

3.5 PROJECT ONE – SORTING LEGOS

- Scratch sorts the blocks based on how they affect the program
- In this project we are going to sort Legos based on certain attributes
- There will be pictures of Legos in a different powerpoint
- Each slide will have a prompt
  - Arrange the Legos according to the prompt

ALPR Module 4. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 4. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of this CS SOL competency across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Classification.pdf
(https://canvas.odu.edu/courses/185316/files/44845402/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845402/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 4. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
K.4 The student will categorize a group of items based on one attribute or the action of each item, with or without a computing device.	English: K.7 (use adjectives to describe attributes) Mathematics: K.9 (comparing attributes); K.12 (sort and classify) Science: K.3 (physical properties); K.6 (senses); K.6 (classifying living and nonliving) Social Studies: K.1 (classify information); K.3 (physical properties); K.4 (describe locations using positional words)
1.5 The student will categorize a group of items based on one or two attributes or the actions of each item, with or without a computing device.	English: 1.7c (sorting words into categories, defining words by attributes) Mathematics: 1.13 (sort and classify concrete objects into appropriate subsets (categories) based on one or two attributes) Science: 1.1c (classify objects); 1.3a (classify objects based on physical properties and explain how the objects were classified); 1.4c (plants can be classified); 1.5c (animals can be classified)
2.5 The student will compare and contrast a group of items based on the attributes or actions of each item, with or without a computing device.	Mathematics: 2.13 (identify and describe solid and plane figures based upon their characteristics) Science: 2.3b (describing characteristics of matter); 2.6a (characteristics of weather) Social Studies:
3.5 The student will compare and contrast a group of items based on attributes or actions classified into at least two sets and two subsets.	English: 3.4 (identifying and classify words based on characteristics) Mathematics: 3.12b (classifying polygons by their attributes) Science: 3.5a (classifying components of an ecosystem) Social Studies: 3.1e (classifying ancient cultures by their attributes)

Computer Science Standard	Opportunity for Integration
4.5 The student will classify and arrange a group of items based on the attributes or actions.	English: 4.4b (classifying words by their attributes) Mathematics: 4.11 (characteristics of plane and solid figures) Science: 4.3d (classification of organisms based upon physical characteristics); 4.5b (classify planets as terrestrial or gas giants)

ALPR Module 4. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- **https://www.acs.org/content/dam/acsorg/education/k-8/inquiry-in-action/second-grade/g2-l1.1-classifying-objects.pdf (https://www.acs.org/content/dam/acsorg/education/k-8/inquiry-in-action/second-grade/g2-l1.1-classifying-objects.pdf)**
- **Raw Data, Classification of Data and Variables: Concepts and Examples (toppr.com) (https://www.toppr.com/guides/economics/organisation-of-data/raw-data-classification-of-data-and-variables/)**
- **30 Ideas to Practice Classification for Kids | Inspiration Laboratories (https://inspirationlaboratories.com/ideas-to-practice-classification-for-kids/#:~:text=30%20Ideas%20to%20Practice%20Classification%20for%20Kids%201,my%20favorite%20is%20the%20Habitat%20Sorting%20activity.%20)**
- **What is an Attribute? - {Definition, Facts & Example} (splashlearn.com) (https://www.splashlearn.com/math-vocabulary/geometry/attribute)**

Published | Preview | Assign To | Edit

ALPR Module 4 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	99
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

Due	For	Available from	Until
-	Everyone	-	-

ALPR Module 4 Assessment

This is a preview of the published version of the quiz

Started: Sep 19 at 5:09pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

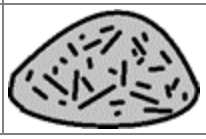
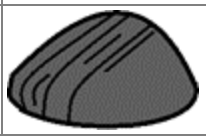
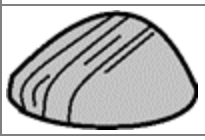
Question 1 33 pts

Ms. Colores has a collection of rocks that she uses in her geology class shown below. The rocks are either rough (spotted) or smooth (streaky) and are bright, grey, or dark.

Ms. Colores wants the rocks to be arranged, listed in order of priority:

1. Rough BEFORE smooth
2. Bright BEFORE grey
3. Grey BEFORE dark

Help Ms. Colores arrange the below set of rocks by ordering them 1st through 5th in the arrangement describe above.

1st	2nd	3rd	4th	5th
				

○ 2nd, 4th, 1st, 5th, 3rd

○ 1st, 5th, 3rd, 2nd, 4th

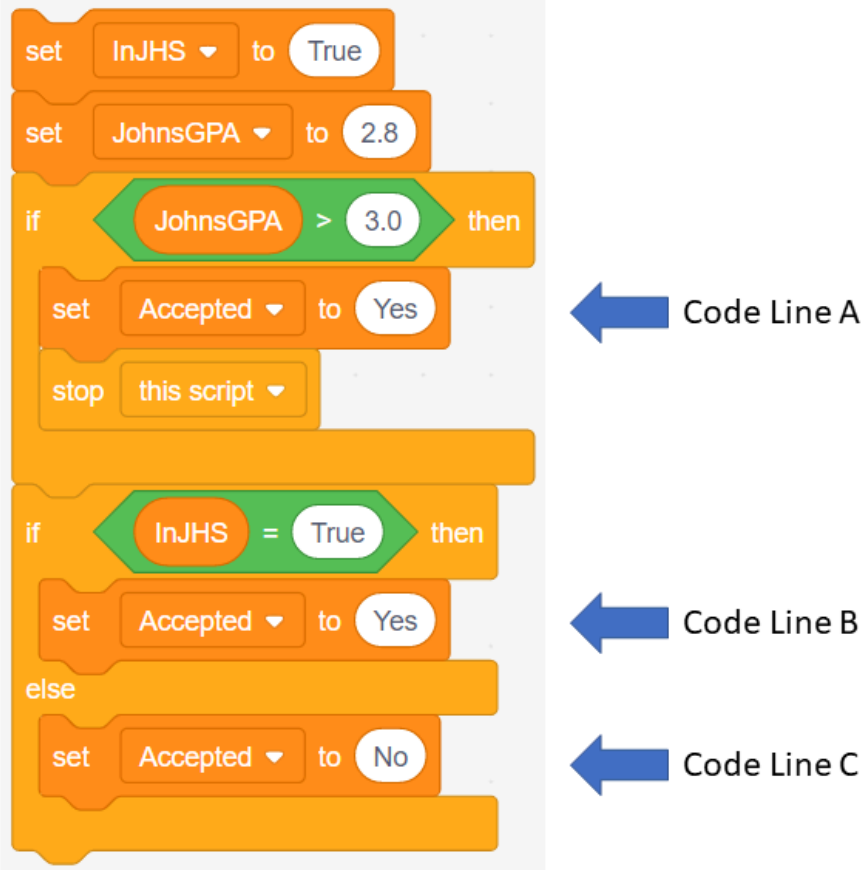
4th, 2nd, 1st, 3rd, 5th

2nd, 4th, 3rd, 5th, 1st

Question 2 33 pts

As a college recruiter, you immediately accept students with a GPA that is higher than a 3.0. Students not meeting that are accepted if they were in the Junior Honor Society.

John has a GPA of 2.8 and was in the Honor Society. In what code line (A, B, or C below) is the decision made on John's acceptance or rejection?



Code Line A

Code Line B

Code Line C

Question 3 33 pts

What is the MOST accurate way to distinguish between an apple and a banana?

Price

Freshness

Shape

Smell

Not saved Submit Quiz

Return to Course Materials

Congratulations, you have completed this module! **Click here to return to Course Materials (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)** .

Algorithms and Programming Module 5

This learning module will focus on **Debugging and Fixing**.

When you complete this learning module, you should be able to:

- **Describe** how an algorithm didn't work (e.g., character is not moving as intended)
- **Analyze** a flawed algorithm to determine possible solutions.
- **Implement** a proposed adjustment to an algorithm that wasn't working as intended.

ALPR Module 5. Teacher Competencies and Alignment with CS SOLs

This learning module includes one teacher competency (in bold), which is aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 7. Describe how an algorithm didn't work (e.g., character is not moving as intended).

ALPR Teacher Competency 8. Analyze a flawed algorithm to determine possible solutions.

ALPR Teacher Competency 9. Implement a proposed adjustment to an algorithm that wasn't working as intended.

- CS 1.3. The student will analyze, correct, and improve (debug) an algorithm that includes sequencing.
- CS 2.3. The student will analyze, correct, and improve (debug) an algorithm that includes sequencing and simple loops, with or without a computing device.
- CS 3.3. The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, and loops.
- CS 4.3. The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops and variables.
- CS 5.3. The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops, conditionals, and variables.

Debugging and Fixing

View the lesson video first, then view the activity video. A PDF of the slides and a transcript of the videos is also included for accessibility.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

3.10 DESIGN A PROGRAM

Why IS IT IMPORTANT?

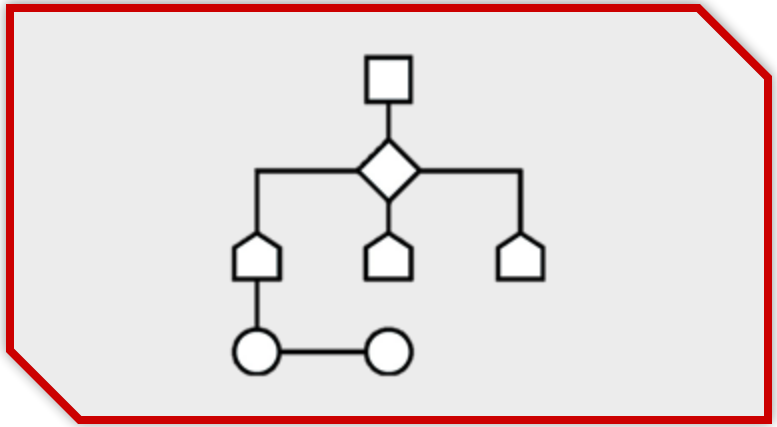
In previous competencies we LISTED the steps of our program.

However, designing requires mapping decisions that can be made.

Learning to use a **planning tool** helps in the design process.

Many design tools exist - which you use is up to you!

In this lesson, we will plan our program by building a *flowchart*



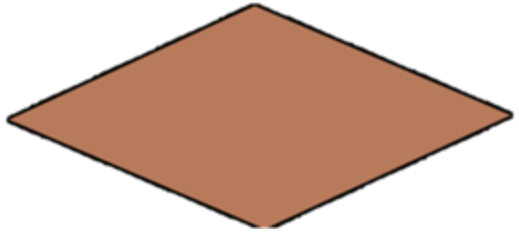
Writing Out a Sequence (Algorithm) AS A FLOWCHART

There's **no official shape and color** for mapping each **flowchart action.**
However, we will use these for **this lesson:**

- Starting point

- Input/Output

- Processing



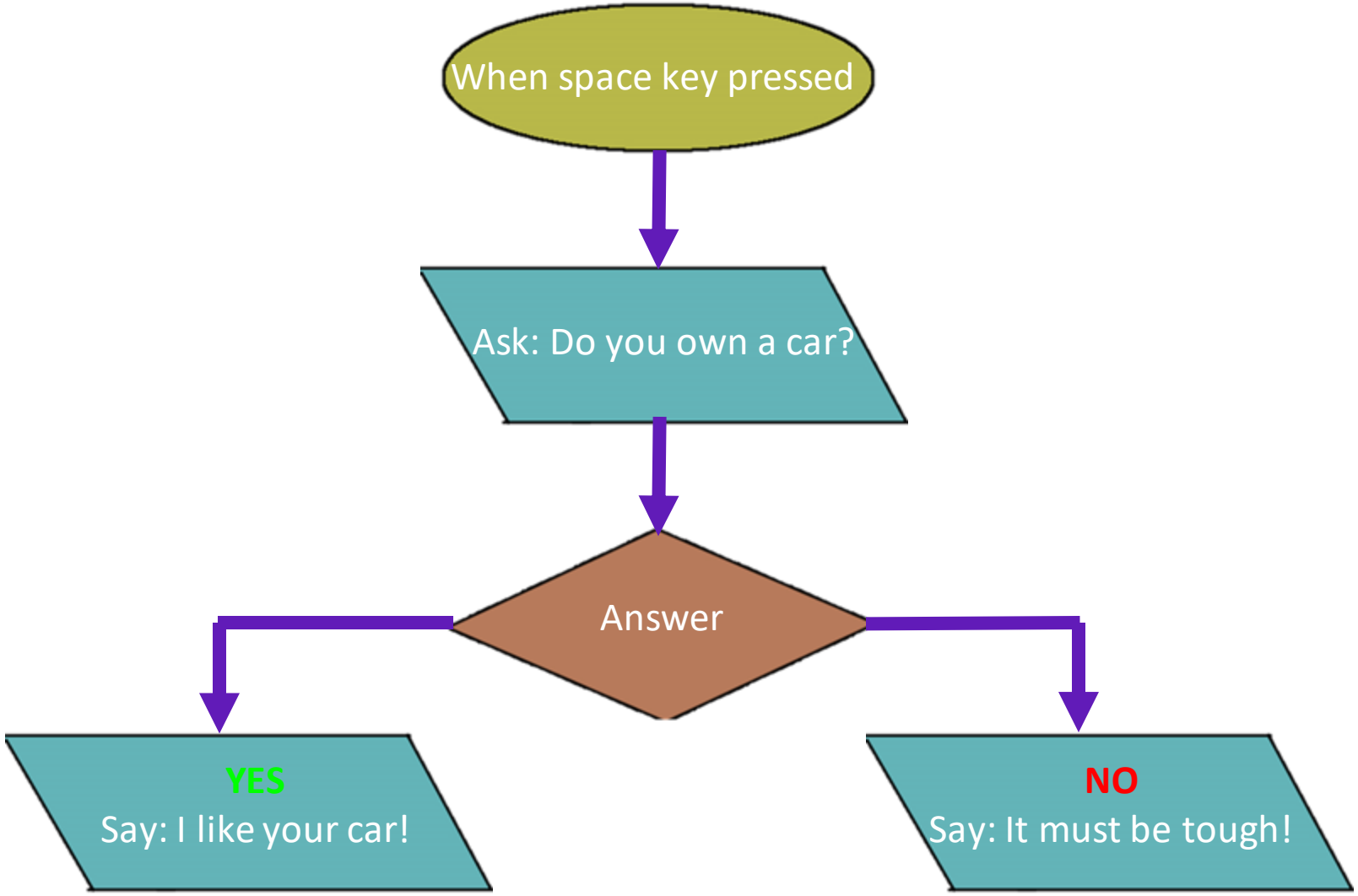
- Decision

- Connectors

The MEANING of our BLOCKS

- **Starting point (oval):** **Start**, **Stop**, and **Halt** in a program's logic flow
  Pause/halt is generally used under `error` conditions.
- **Input/Output (parallelogram):** denotes any function of **input**/**output**
  Input from input devices or output on output devices are indicated with this.
- **Processing (rectangle):** **arithmetic** instructions (**i.e. +, -, *, /**)
- **Decision (diamond):** **yes**/**no** question or **true**/**false** decision points
- **Connectors (circle):** useful connector to **avoid confusion** for complex charts

EXAMPLE of an ALGORITHM FLOWCHART



Algorithm Flowchart Creation Project

The Goal: a program that adds or subtracts two numbers as desired by the user

How it Should Work: **Starting** when the **green flag** is clicked,
it **asks** whether to perform **addition** or **subtraction**.
It then **asks** what the **first number** it adds or subtracts with is,
followed by the **second number**.
Finally, it **displays** the **result** of the performed arithmetic.

NOTE:

- You may look at the Scratch blocks to ***plan ahead***, but use a planning tool **before moving any blocks!**
- You can define you own ***shape*** and ***color*** for each flowchart action, but **keep it consistent.**

Slide 1

Hello, everyone, and welcome to the ARCS Microcredential of Algorithms and Programming. This is the Debugging section. We're going to talk about debugging programs today.

Slide 2

First, let's step into what a bug is. A bug is when your program doesn't function as planned – whenever something's wrong. The more complicated your program, the more likely that bugs appear in your code. This will always remain true. No matter the level you get to, you always will find bugs. The more experience you have, the less likely there are to be bugs in your program. Bugs could cause other bugs, which then would make finding the original bug even more difficult. Because every single step is important in a program, when one step messes up, it could cause other steps to mess up as well!

Slide 3

We now will cover how to find bugs. When debugging, knowing the desired output for the program will make it easier to read through the code and find any bugs. Generally speaking, knowing what you're expecting makes it easier to analyze the situation - this is essentially the same with bugs. Once you identify the symptoms of your bugs, you can narrow down where the bug may be hiding. For example, you could be expecting an output after an if-statement; the issue is your if-statement is not saying something that you wanted to say. That problem can be solved by looking at that if-statement and seeing if that boolean is true or false.

Slide 4

Now it's your turn. What we're going to do is complete the scratch debugging activity together. If you want, you can complete the unplugged "define-a-dance" activity on your own. Let's go ahead and get into that in the next video.

Defining a Dance

In this activity you will choose a dance from the options. Then you will write, in your own words, in as detailed steps as possible, how to do the chosen dance move. You do not need to do this in a flowchart style. However, make sure you are clear with the start of your dance and the steps that follow until the end. Something that makes this activity very fun is when you give these instructions to someone you know without telling them what dance you tried to replicate. Make sure to inform them they need to follow your directions exactly as written.

Dance moves: The Moonwalk | The Robot | The Dougie | The Twist | The Floss

Challenge: After writing the detailed steps, rewrite them in order to debug any errors

DEBUGGING IN SCRATCH

Slide 1
Hello, everyone, and welcome to the ARCS Microcredential of Algorithms and Programming. This is the Debugging section. We're going to talk about debugging programs today.

INSTRUCTIONS

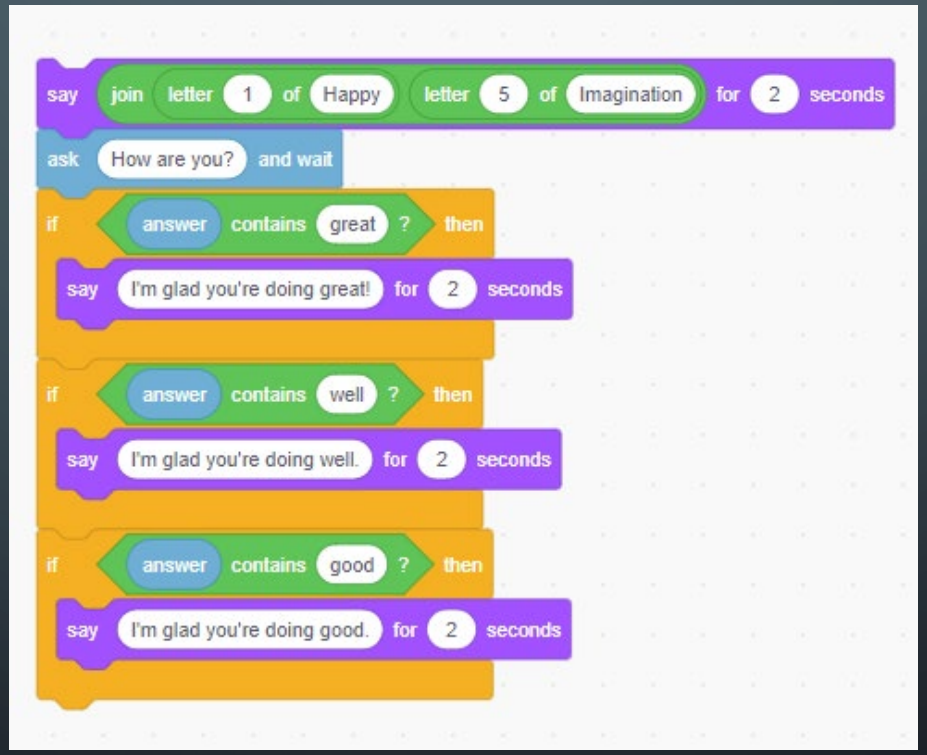
- In this activity there are two programs with errors
- Work through the three programs and make them work
- On the slides with the pictures of the code will be a link to the programs
- After finding the errors see if you can make the code run properly

Slide 2

First, let's step into what a bug is. A bug is when your program doesn't function as planned – whenever something's wrong. The more complicated your program, the more likely that bugs appear in your code. This will always remain true. No matter the level you get to, you always will find bugs. The more experience you have, the less likely there are to be bugs in your program. Bugs could cause other bugs, which then would make finding the original bug even more difficult. Because every single step is important in a program, when one step messes up, it could cause other steps to mess up as well!

HOW ARE YOU - ERROR

GOAL: When the green flag is clicked, say a greeting and then ask how the user is doing. Lastly respond accordingly to the users response.



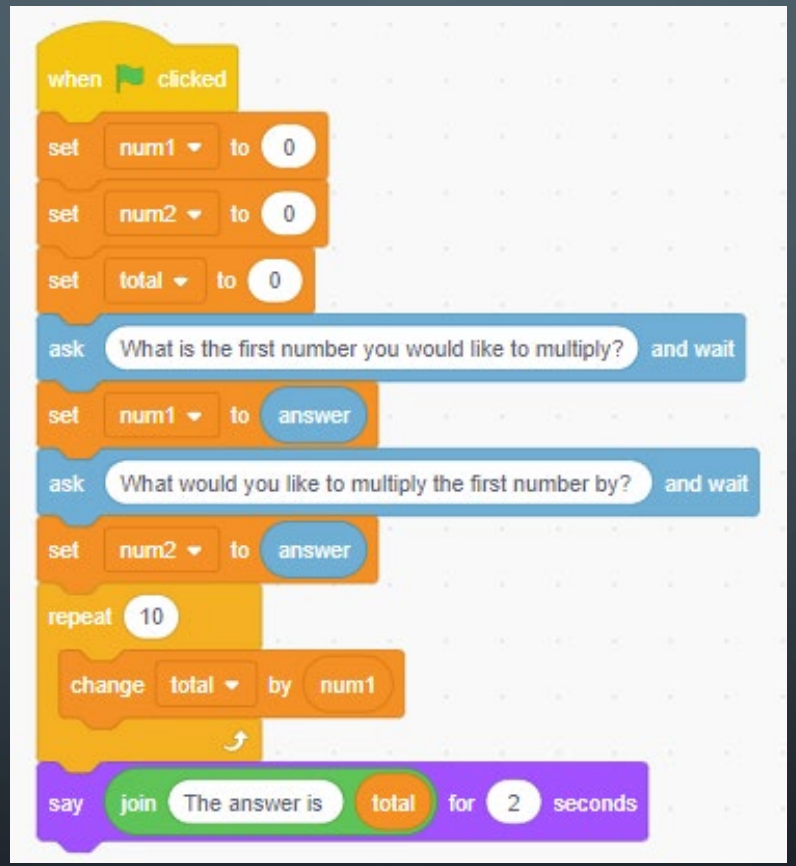
Link: https://scratch.mit.edu/projects/547614351

Slide 3

We now will cover how to find bugs. When debugging, knowing the desired output for the program will make it easier to read through the code and find any bugs. Generally speaking, knowing what you're expecting makes it easier to analyze the situation - this is essentially the same with bugs. Once you identify the symptoms of your bugs, you can narrow down where the bug may be hiding. For example, you could be expecting an output after an if-statement; the issue is your if-statement is not saying something that you wanted to say. That problem can be solved by looking at that if-statement and seeing if that boolean is true or false.

CALCULATOR - ERROR

GOAL: When the green flag is clicked, take the first number, then the number to multiply the first number by. Lastly output the product.



Link: https://scratch.mit.edu/projects/547649111

Slide 4
Now it's your turn. What we're going to do is complete the scratch debugging activity together. If you want, you can complete the unplugged "define-a-dance" activity on your own. Let's go ahead and get into that in the next video.

ALPR Module 5. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 5. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of this CS SOL competency across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Debugging and Fixing.pdf
(https://canvas.odu.edu/courses/185316/files/44845403/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845403/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 5. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
1.3 The student will analyze, correct, and improve (debug) an algorithm that includes sequencing.	English: 1.6e (reread and self-correct); Mathematics: 1.14 (growing and repeating patterns)
2.3 The student will analyze, correct, and improve (debug) an algorithm that includes sequencing and simple loops, with or without a computing device.	English: 2.11 (edit writing for proper use of capitalization, punctuation, and spelling) Mathematics: 2.16 (patterns)
3.3 The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, and loops.	English: 3.9 (editing grammar and spelling resembles the process of debugging) Mathematics: 3.3b (solving multistep problems)
4.3 The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops and variables.	Mathematics: 4.4-4.6 (problem solving techniques) Science: 4.1c (analyze data from tests of an object or tool to determine if it works as intended)
5.3 The student will analyze, correct, and improve (debug) an algorithm that includes sequencing, events, loops, conditionals, and variables.	English: 5.7l (revising text for clarity and grammar) Mathematics: 5.4-5.7 (solving problems, using order of operations); 5.18 (number patterns) Science: 5.1c (use data to evaluate and refine design solutions; 5.1d (generate and compare multiple solutions to problems based on whether they meet criteria and constraints)

ALPR Module 5. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- https://www.acs.org/content/dam/acsorg/education/k-8/inquiry-in-action/second-grade/g2-l1.1-classifying-objects.pdf (https://www.acs.org/content/dam/acsorg/education/k-8/inquiry-in-action/second-grade/g2-l1.1-classifying-objects.pdf)
- What are the two types of Errors in Computer Programming ? | Penjee, Learn to Code (https://blog.penjee.com/what-are-the-2-types-of-errors-in-programming/#:~:text=%202%20General%20Categories%20of%20Computer%20Programming%20Error,of%20a%20logic%20error%20in%20Penjee.%20More%20)
- Different Types of Errors in Programming Languages | CodingAlpha (https://www.codingalpha.com/types-of-errors-in-programming/)
- Thinking About Errors in Your Code Differently (codecademy.com) (https://news.codecademy.com/errors-in-code-think-differently/)
- Intro to Coding - Testing and Fixing Code Flashcards | Quizlet (https://quizlet.com/539749273/intro-to-coding-testing-and-fixing-code-flash-cards/)
- How to try to fix your own broken code (codewith.mu) (https://codewith.mu/en/howto/1.0/fix_code)
- Techniques to Troubleshoot Your Code - YouTube (https://www.youtube.com/watch?v=YYrUVHORcc8)
- 10 Debugging Tips for Beginners: How to Troubleshoot and Fix Your Code Without Pulling Your Hair Out (hartleybrody.com) (https://blog.hartleybrody.com/debugging-code-beginner/#:~:text=10%20Debugging%20Tips%20for%20Beginners%3A%20How%20to%20Troubleshoot,implementation.%20...%2010%20How%20to%20ask%20for%20help)
- 7 Steps to Debug Efficiently and Effectively | Codementor (https://www.codementor.io/@mattgoldspink/how-to-debug-code-efficiently-and-effectively-du107u9jh)

Published | Preview | Assign To | Edit

ALPR Module 5 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	99
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

Due	For	Available from	Until
-	Everyone	-	-

ALPR Module 5 Assessment

This is a preview of the published version of the quiz

Started: Sep 19 at 5:16pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Question 1 33 pts

The Scratch program determines if an order is charged a $5.00 delivery fee by checking if the variable Order has the value “DELIVERY.”

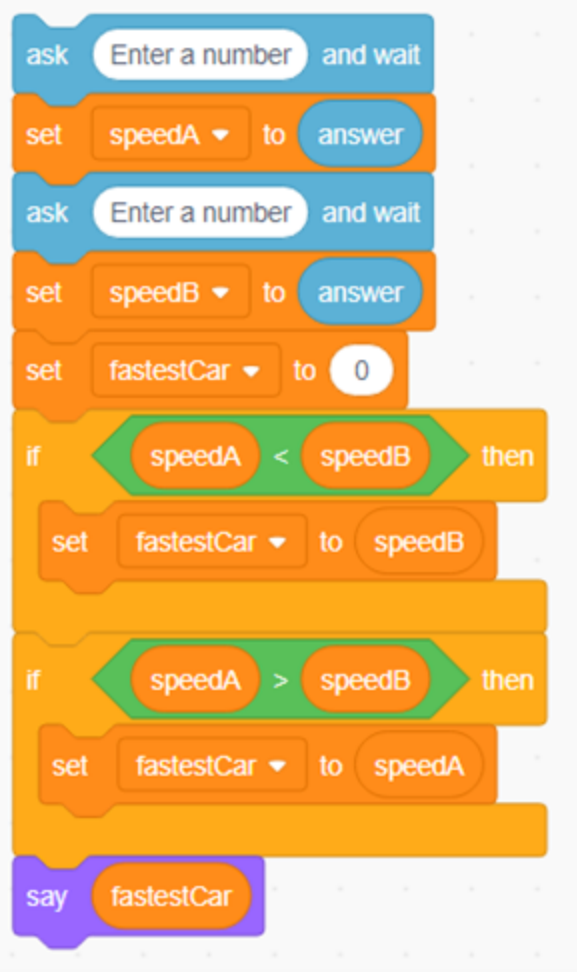
Does this program output the expected result? Why?

- No, Order should be “PICKUP” instead of "DELIVERY"
- No, there needs to be a loop in the program.
- No, the variable DeliveryFee is used but is never given a value.

Yes, this program outputs the intended result.

Question 2 33 pts

In a hurry when writing her program, Alice switches statements 4 and 5 of the program. So, her program becomes:



This accidental swapping doesn't impact the outcome of the program (i.e., the value displayed by the two programs) is the same no matter what the values of x and y are.

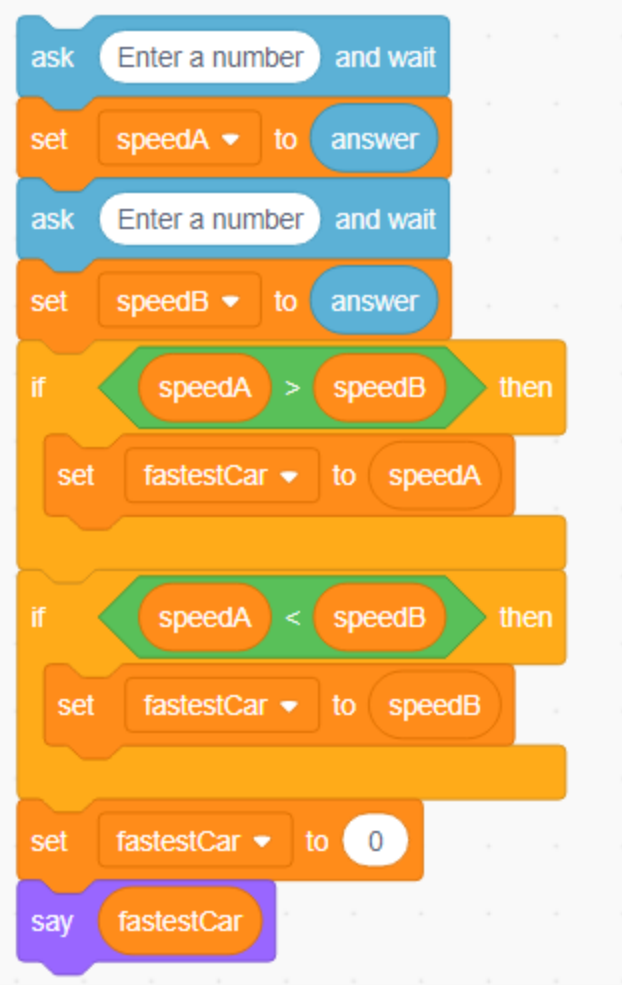
True

False

Question 3 33 pts

Bob is helping Alice with her computer code comparing the speeds of two cars. Also in a hurry, Bob places the third step of the program after the conditional statements. His algorithm is presented below.

True or false: This accidental swapping doesn't impact the outcome of the program (i.e. the value displayed by the two programs is the same) no matter what the values of x and y are.



○ True

○ False

Not saved Submit Quiz

Return to Course Materials

Congratulations, you have completed this module! **Click here to return to Course Materials (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)** .

Algorithms and Programming Module 6

This learning module will prepare you to **Plan and Decompose**.

When you complete this learning module, you should be able to:

- **Design** a program using a planning tool.
- **Describe** how an iterative design process can improve an algorithm.
- **Analyze** and **decompose** a problem into subproblems.
- **Explain** why multiple smaller problems may be easier to solve than one large problem.

ALPR Module 6. Teacher Competencies and Alignment with CS SOLs

This learning module includes one teacher competency (in bold), which is aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 10. Design a program using a planning tool.

ALPR Teacher Competency 11. Describe how an iterative design process can improve an algorithm.

ALPR Teacher Competency 12. Analyze and decompose a problem into subproblems.

ALPR Teacher Competency 13. Explain why multiple smaller problems may be easier to solve than one large problem.

- CS K.3. The student will create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer).
- CS 1.4. The student will plan and create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, sequential graphic organizer).
- CS 2.4. The student will plan and create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, sequential graphic organizer).
- CS 3.4. The student will create a plan as part of the iterative design process, independently and/or collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).
- CS 3.6. The student will break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively.
- CS 4.4. The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).
- CS 4.6. The student will break down (decompose) a larger problem into smaller sub-problems, both independently and collaboratively.
- CS 5.4. The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).
- CS 5.5. The student will break down (decompose) a larger problem into smaller sub-problems, both independently and collaboratively.

Plan and Decompose

View the lesson video first, then view the activity video. A PDF of the slides and a transcript of the videos is also included for accessibility.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

3.10 DESIGN A PROGRAM

Slide 1
Hello, everyone, and welcome to the ARCS Microcredentials Algorithms and Programming. In this session, we are going to design a program using a planning tool.

Why IS IT IMPORTANT?

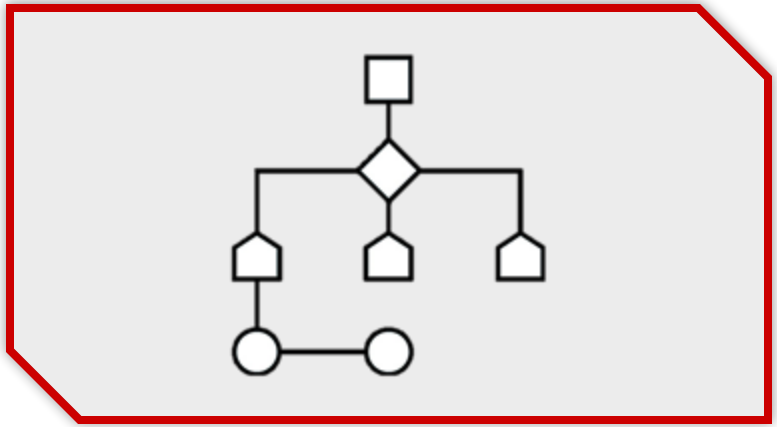
In previous competencies we LISTED the steps of our program.

However, designing requires mapping decisions that can be made.

Learning to use a **planning tool** helps in the design process.

Many design tools exist - which you use is up to you!

In this lesson, we will plan our program by building a *flowchart*



Slide 2

You might be wondering why it is important to design using a planning tool. In the previous competencies, we listed the steps of our program. However, designing requires mapping decisions that can be made, like having an understanding of what your program can and will do. Learning to use a planning tool helps in the design process, and many design tools exist. So whichever you use is up to you.

In this lesson, we will plan our programming by building a flow chart, and it looks somewhat similar to this, where this would be our start point. This would be a decision. This would be different possibilities after our decision is made, and this would be something that follows afterward. Planning out your program can be beneficial, as it can help prevent errors in having an expectation before you go into programming your project.

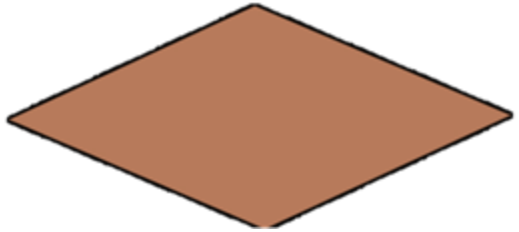
Writing Out a Sequence (Algorithm) AS A FLOWCHART

There's **no official shape and color** for mapping each **flowchart action.**
However, we will use these for **this lesson:**

- Starting point

- Input/Output

- Processing



- Decision

- Connectors

Slide 3

With writing out a sequence algorithm as a flow chart, there's no official shape and color for mapping each flow chart action. However, we will use these for this lesson. So we'll use this Oval as a starting point. We're going to use parallelograms as input output. The processing rectangle is for running operations (or, in other words, tasks with operators). We're going to use a diamond as a decision statement and circles for our connectors.

The MEANING of our BLOCKS

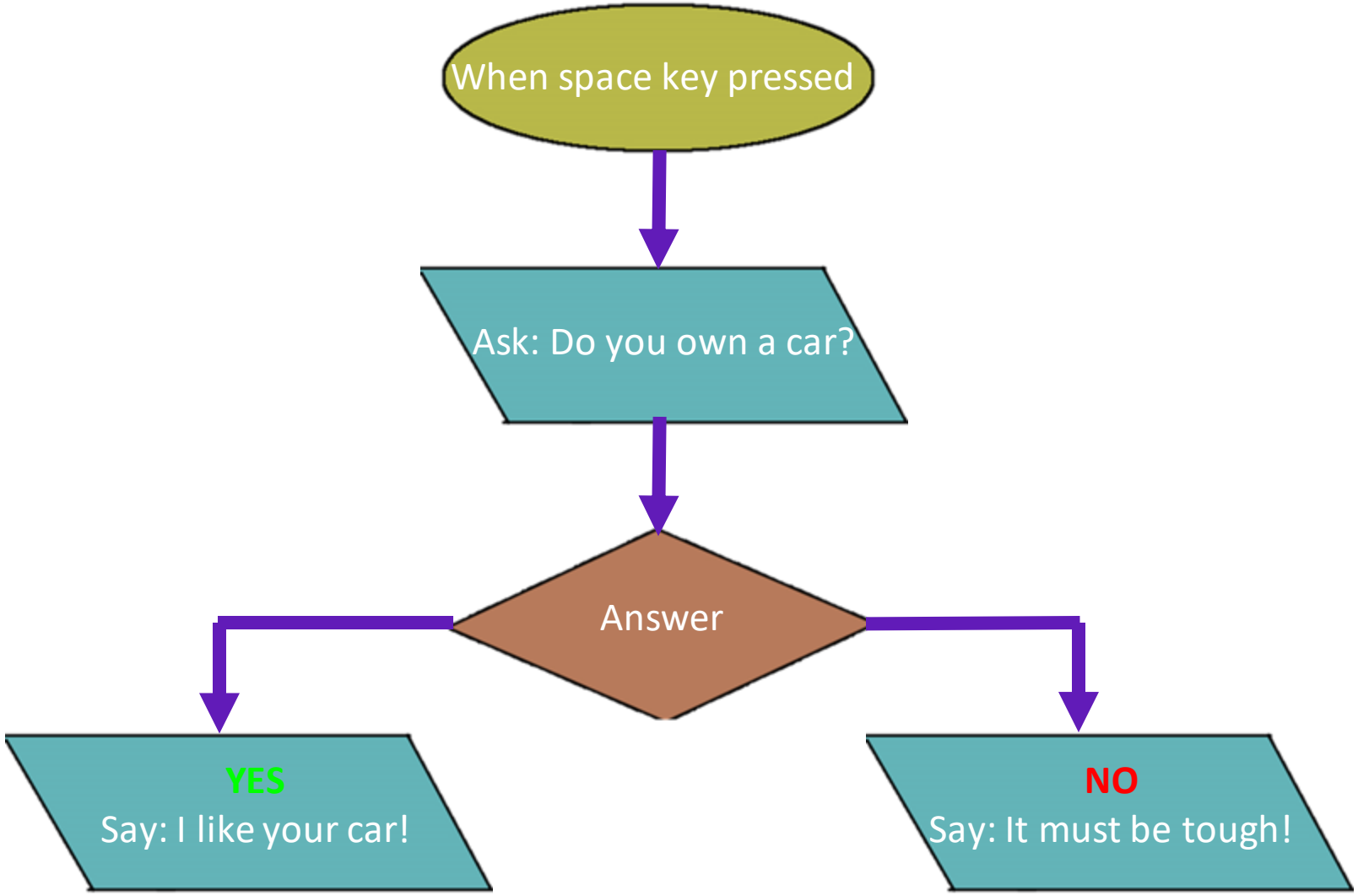
- **Starting point (oval):** **Start**, **Stop**, and **Halt** in a program's logic flow
  Pause/halt is generally used under `error` conditions.
- **Input/Output (parallelogram):** denotes any function of **input**/**output**
  Input from input devices or output on output devices are indicated with this.
- **Processing (rectangle):** **arithmetic** instructions **(i.e. +, -, *, /)**
- **Decision (diamond):** **yes**/**no** question or **true**/**false** decision points
- **Connectors (circle):** useful connector to **avoid confusion** for complex charts

Slide 4
We will now go into the meaning of our blocks. We have our starting point, which is the oval. It signifies the start, stop and halt in the program logic flow. Pause or halt are only used under error conditions, like we did in our Simon Says program. We needed to stop it so that it didn't continue on to the next path that it didn't need to run.

The input/output -or the parallelogram -denotes any function of input or output. So input from input devices or output from output devices are indicated with this like it. An example of that would be when we ask when we use the ask block and we get an answer back. That would be an input or an output would be the say blockThe processing (rectangle) is for arithmetic instructions: addition, subtraction, multiplication, division, greater than, less than, etc. These are the operator blocks that we learned about in the previous section. The decision block would be a yes or no question or true or false decision points. Or "if this, then" things of that sort.

Connectors (circle) make for useful connections to avoid confusion for comp lex charts. So an example of that, which we're not going to use these a lot, is the capability in scratch for you to write a script. Depending on how that script plays out, you can activate another script or program or algorithm. We're not going to really go into this too much, but if you feel like exploring that further, feel free.

EXAMPLE of an ALGORITHM FLOWCHART



Slide 5

Here's an example of an algorithm flow chart. Our event (or our starting point) is when the space key is pressed. Here's our input block. Ask, "Do you own a car?", then get the answer. If the answer is "Yes", say, "I like your car". If the answer is "No", say, "it must be tough". I know for one that I quite enjoy driving my car -it makes things a lot easier.

Algorithm Flowchart Creation Project

The Goal: a program that adds or subtracts two numbers as desired by the user

How it Should Work: **Starting** when the **green flag** is clicked,
it **asks** whether to perform **addition** or **subtraction**.
It then **asks** what the **first number** it adds or subtracts with is,
followed by the **second number**.
Finally, it **displays** the **result** of the performed arithmetic.

NOTE:

- You may look at the Scratch blocks to ***plan ahead***, but use a planning tool **before moving any blocks!**
- You can define you own ***shape*** and ***color*** for each flowchart action, but **keep it consistent.**

Slide 6
Let's go ahead and make our own. We'll go over the solution in the next video. Time for the Algorithm Flowchart Creation Project.

Slide 1

Hello, everyone, and welcome to the ARCS Microcredentials Algorithms and Programming. In this session, we are going to design a program using a planning tool.

Slide 2

You might be wondering why it is important to design using a planning tool. In the previous competencies, we listed the steps of our program. However, designing requires mapping decisions that can be made, like having an understanding of what your program can and will do. Learning to use a planning tool helps in the design process, and many design tools exist. So whichever you use is up to you.

In this lesson, we will plan our programming by building a flow chart, and it looks somewhat similar to this, where this would be our start point. This would be a decision. This would be different possibilities after our decision is made, and this would be something that follows afterward. Planning out your program can be beneficial, as it can help prevent errors in having an expectation before you go into programming your project.

Slide 3

With writing out a sequence algorithm as a flow chart, there's no official shape and color for mapping each flow chart action. However, we will use these for this lesson. So we'll use this Oval as a starting point. We're going to use parallelograms as input output. The processing rectangle is for running operations (or, in other words, tasks with operators). We're going to use a diamond as a decision statement and circles for our connectors.

Slide 4

We will now go into the meaning of our blocks. We have our starting point, which is the oval. It signifies the start, stop and halt in the program logic flow. Pause or halt are only used under error conditions, like we did in our Simon Says program. We needed to stop it so that it didn't continue on to the next path that it didn't need to run.

The input/output - or the parallelogram - denotes any function of input or output. So input from input devices or output from output devices are indicated with this like it. An example of that would be when we ask when we use the ask block and we get an answer back. That would be an input or an output would be the say block.

The processing (rectangle) is for arithmetic instructions: addition, subtraction, multiplication, division, greater than, less than, etc. These are the operator blocks that we learned about in the previous section.

The decision block would be a yes or no question or true or false decision points. Or "if this, then" - things of that sort.

Connectors (circle) make for useful connections to avoid confusion for complex charts. So an example of that, which we're not going to use these a lot, is the capability in scratch for you to write a script. Depending on how that script plays out, you can activate another script or program or algorithm. We're not going to really go into this too much, but if you feel like exploring that further, feel free.

Slide 5

Here's an example of an algorithm flow chart. Our event (or our starting point) is when the space key is pressed. Here's our input block. Ask, "Do you own a car?", then get the answer. If the answer is "Yes", say, "I like your car". If the answer is "No", say, "it must be tough". I know for one that I quite enjoy driving my car - it makes things a lot easier.

Slide 6

Let's go ahead and make our own. We'll go over the solution in the next video. Time for the Algorithm Flowchart Creation Project.

ALPR Module 6. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 6. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of this CS SOL competency across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Plan and Decompose.pdf
(https://canvas.odu.edu/courses/185316/files/44845371/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845371/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 6. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
K.3 The student will create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer).	English: K.2 (tell stories orally); K.8 (sequence stories using beginning, middle, and end); K.11 (write in a variety of forms) Social Studies: K.3 (sequence events)
1.4 The student will plan and create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, sequential graphic organizer).	English: 1.12b (use prewriting strategies) Mathematics: 1.3 (ordinal position)
2.4 The student will plan and create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, sequential graphic organizer).	English: 2.10c (using prewriting strategies) Mathematics: 2.3 (ordinal numbers through 20) Social Studies: 2.1f (create flow charts to show change in technology over time)
3.4 The student will create a plan as part of the iterative design process, independently and/or collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 3.8c (use a variety of prewriting strategies to plan and organize writing) Science: 3.1a (define a simple problem); 3.1b (use tools and/or materials to design and/or build a device to solve a specific problem); 3.1c (analyze data from tests of an object or tool to determine if it works as intended); 3.1f (communicate design ideas and/or solutions with others)
3.6 The student will break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively.	Mathematics: 3.3b (breaking down multi-step problems) Science: 3.1a (defining problems)
4.4 The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 4.7d (using prewriting strategies) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system); 4.1b (use tools and/or materials to design and/or build a device that solves a specific problem); 4.1c (analyze data from tests of an object or tool to determine if it

Computer Science Standard	Opportunity for Integration
	works as intended); 4.1e (identify limitations of a model); 4.1f (communicate design ideas or solutions to others)
4.6 The student will break down (decompose) a larger problem into smaller sub-problems, both independently and collaboratively.	English: 4.4a,b (break down sentences to fix grammar) Mathematics: 4.4b (strategies for determining sum/difference/product of two whole numbers) Science: 4.1a (define a simple design problem that can be solved through the development of an object, tool, process or system)
5.4 The student will create a plan as part of the iterative design process, both independently and collaboratively using strategies such as pair programming (e.g., storyboard, flowchart, pseudo-code, story map).	English: 5.1 (preparing prewriting tools); 5.7c (using a variety of prewriting tools) Science: 5.1a (defined design problems that can be solved through the development of an object, tool, process, or a system); 5.1b (planning investigations and writing procedures); 5.1c (use data to evaluate and refine design solutions); 5.1d (generate and compare multiple solutions to problems based on whether they meet criteria and constraints); 5.1e (identify limitations of models); 5.1f (communicate design ideas or solutions to others)
5.5 The student will break down (decompose) a larger problem into smaller sub-problems, both independently and collaboratively.	English: 5.1 (working together on presentations, splitting up tasks); 5.3b (deconstructing a media message into its components) Mathematics: 5.4 (distributive property); 5.6 (multi-step problems) Science: 5.1c (breaking down data sets to reveal patterns); 5.3c (identifying all forces at work on an object)

ALPR Module 6. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- Four Steps to Take before Writing a Computer Program - dummies (https://www.dummies.com/programming/four-steps-to-take-before-writing-a-computer-program/#:~:text=%20The%20following%20four%20steps%20are%20crucial%20to,it%20a%20Windows%20computer%2C%20a%20Macintosh%2C...%20More%20)
- Planning a programming project (article) | Khan Academy (https://www.khanacademy.org/computing/computer-programming/programming/good-practices/a/planning-a-programming-project)
- How to Create a Program (with Pictures) - wikiHow (https://www.wikihow.com/Create-a-Program)
- Chapter 01 Planning Computer Program (re-upload) (slideshare.net) (https://www.slideshare.net/bluejayjunior/chapter-01-planning-computer-program-reupload)
- 6.4. Iteration Improves the Turtle Program — LaunchCode's LCHS documentation (https://education.launchcode.org/lchs/chapters/loops/turtle-iteration.html)
- Iterative Loops (stanford.edu) (https://web.stanford.edu/group/sisl/k12/optimization/MO-unit1-pdfs/1.8iterativeloops.pdf)
- Programming – The Purpose Of Loops – DPS Computing (https://www.dpscomputing.com/blog/2012/09/13/programming-the-purpose-of-loops/#:~:text=The%20purpose%20of%20loops%20is%20to%20repeat%20the,be%20dictated%20by%20a%20certain%20condition%20being%20met.)
- (4) Why do we use looping in programming? - Quora (https://www.quora.com/Why-do-we-use-looping-in-programming)
- Looping code - Learn web development | MDN (mozilla.org) (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code)
- What is decomposition? - BBC Bitesize (https://www.bbc.co.uk/bitesize/topics/zkcqn39/articles/z8ngr82)
- L05-DecompositionAbstraction.pdf (toronto.edu) (http://www.cs.toronto.edu/~sme/CSC444F/slides/L05-DecompositionAbstraction.pdf)
- What Is An Algorithm and Why Are They Important (mycodingplace.com) (https://www.mycodingplace.com/post/what-is-an-algorithm-and-why-are-they-important)

Published | Preview | Assign To | Edit

ALPR Module 6 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 75 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	100
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

Due	For	Available from	Until
-	Everyone	-	-

ALPR Module 6 Assessment

This is a preview of the published version of the quiz

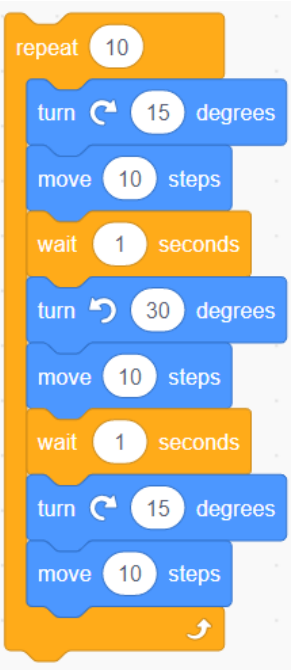
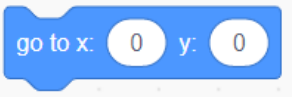
Started: Sep 19 at 5:25pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 75 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Question 1 25 pts

What order should the following code segments go in to replicate the slithering snake program?

1	
2	
3	

○ 1,2,3

○ 2,3,1

○
3,2,1

○
1,3,2

Question 2 25 pts

Select the statement below that reflects actions that could be considered an example of improving an algorithm.

○
Carrying each grocery item out of the bag when unloading your car.

○
Rinsing off your toothbrush before applying toothpaste.

○
Taking an alternative route that takes less time to reach the destination.

○
Using a coupon on your purchase.

Question 3 25 pts

When counting her change, Mary groups the coins by their value (i.e. pennies, nickels, dimes, etc.). She calculates the value of each group, then adds the sums in order of coin value.

Calculating the value of each group first is considered a(n) _____.

○
Sequence

○
Subproblem

○
Pattern

○
Attribute

Question 4 25 pts

ABACUS, a computer sold by StoneAge Computer Corporation, can only read, add, and display numbers; however, it can be programmed to allow for repeating these operations.

Charlie, a devoted user of ABACUS, would like to somehow use it to read and multiply two numbers *m* and *n* and display the result. Mark all that are true:

- [ ] Charlie can't accomplish this task because ABACUS can only add numbers, not multiply them.
- [ ] Charlie can use ABACUS to accomplish this task because multiplication can be done using repeated addition.
- [ ] Charlie can accomplish this task by programming ABACUS to repeatedly add m to itself n-1 times.
- [ ] Charlie can accomplish this task by programming ABACUS to repeatedly add n to itself m-1 times.

Not saved Submit Quiz

Return to Course Materials

Congratulations, you have completed this module! **Click here to return to Course Materials (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)** .

Algorithms and Programming Module 7

This learning module will focus on **Giving Credit to Sources**.

When you complete this learning module, you should be able to:

- **Review** a program written by a programmer and identify portions that may have been created by others.
- **Explain** why it is important to give credit to authors.
- **Describe** when it is acceptable to use people's work, and how to give credit to sources.

ALPR Module 7. Teacher Competencies and Alignment with CS SOLs

This learning module includes one teacher competency (in bold), which is aligned with the following Computer Science SOLs (bulleted):

ALPR Teacher Competency 14. Review a program written by a programmer and identify portions that may have been created by others.

ALPR Teacher Competency 15. Explain why it is important to give credit to authors.

ALPR Teacher Competency 16. Describe when it is acceptable to use people's work, and how to give credit to sources.

- CS 1.6. The student will acknowledge that materials are created by others (e.g., author, illustrator).
- CS 2.6. The student will acknowledge that materials are created by others (e.g., author, illustrator, and website).
- CS 3.7. The student will give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects).
- CS 4.7. The student will give credit to sources when borrowing or changing ideas (e.g., using information, pictures created by others, using music created by others, remixing programming projects).
- CS 5.6. The student will give credit to sources when borrowing or changing ideas (e.g., using information, pictures created by others, using music created by others, remixing programming projects).

Giving Credit to Sources

View the video presented here. A PDF of the slides and a transcript of the videos is also included for accessibility.

***We recommend that you right click on the link and choose "Open in New Tab" for best viewing.**

C

GIVING CREDIT

Slide 1
Hello, everyone, and welcome to the final section of the ARCS Microcredential of Algorithms and programming. This one's a really short section. We only have 3 slides, and it's about giving credit where credit is due.

Giving Credit Where it's Due

Any resources you use,

including those found online,

have an **author** or **creator**.

Why it's **important** to give credit:

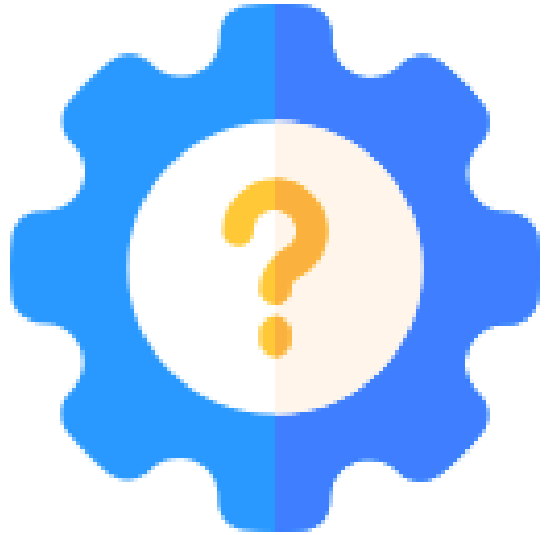
- ✓**Show appreciation** of the work used
- ✓**Validates** research
- ✓**Acknowledges the creator** of the material

Slide 2

It's important to give credit where credit is due. Any resources you use, including those found online, have an author or a creator. Here's why it's important to give credit: it shows appreciation of the work used, it validates research, and it acknowledges the creator of the material. A lot of work has been put into the resources you use, which is likely why you use them for reference, so showing that appreciation is very important.

What Should Be Credited When Used?

- Uncommon information 
- Portions of someone's program 
- Images 
- Music 



Slide 3
What should be credited when used? Information that's uncommon, portions of someone's program (if you go online and find a piece of a program, say "this portion was taken from/ provided by", etc.) Anytime you use a picture or music, give credit where credit is due as well. There's no project with this one, so that concludes this section.

ALPR Module 7. Curricular Alignment and Curriculum Framework

The information contained in this section will help you as you begin to develop your lesson plan for this course, including the following information for this competency:

- Computer Science SOL vertical alignment (K-5)
- Cross-curricular alignment
- Background information and Essential Skills, Questions, and Vocabulary (Curriculum Framework)

ALPR Module 7. CS SOL Vertical Alignment (K-5)

The attached file illustrates the vertical alignment of this CS SOL competency across K-5 grade span. The yellow highlighted areas indicate the CS standards with which this teacher competency align. The light blue shaded areas indicate introductory level skills and the dark blue shaded areas indicate proficiency of the standards.

Please note that this vertical alignment document was developed by TCEP faculty and has not been vetted by the VDOE or CodeVA.

ALPR Vertical Alignment-Crediting Sources.pdf
(https://canvas.odu.edu/courses/185316/files/44845376/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845376/download?download_frd=1)

Computer Science SOLs	Grade					
Algorithms and Programming	K	1	2	3	4	5
Construct sets of step-by-step instructions (algorithms) either independently or collaboratively including sequencing that emphasize the beginning, middle, and end. K.1, 1.1 (+ sequencing and loops), 2.1(+ identifying events), 3.1(+ using events), 4.1 (+ using variables and performing number calculations), 5.1 (+ using conditionals, if/then).						
Construct programs to accomplish tasks as a means of creative expression using a block based programming language or unplugged activities, either independently or collaboratively, including sequencing, emphasizing the beginning, middle, and end. K.2, 1.2 (+ sequencing and loops), 2.2 (+ identifying events), 3.2, 4.2 (+ using variables and performing number calculations), 5.2 (+ using conditionals, if/then).						
Create a design document to illustrate thoughts, ideas, and stories in a sequential (step-by-step) manner (e.g., story map, storyboard, and sequential graphic organizer). K.3, 1.4, 2.4						
Categorize a group of items based on one attribute or the action of each item, with or without a computing device. K.4, 1.5, 2.5 (compare and contrast items), 3.5 (compare and contrast 2 sets and 2 subsets of items), 4.5 (classify and arrange a group of items)						
Analyze, correct, and improve (debug) an algorithm that includes sequencing. 1.3, 2.3, 3.3 (+ events and loops), 4.3 (+ variables), 5.3						
Acknowledge that materials are created by others (e.g., author, illustrator). 1.6, 2.6						
Create a plan as part of the iterative design process, independently and/or collaboratively, using a variety of strategies (e.g., pair programming, storyboard, flowchart, pseudocode, story map). 3.4, 4.4, 5.4						
Break down (decompose) a larger problem into smaller sub-problems, independently or collaboratively. 3.6, 4.6, 5.5						
Give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects). 3.7, 4.7, 5.6						
Light blue – Introduction Dark blue - Proficient						

ALPR Module 7. CS Cross-Curricular Alignment (K-5)

Listed below are some suggested areas of integration from the VDOE, but this is not an exhaustive list. What areas do you see for cross-curricular alignment?

Computer Science Standard	Opportunity for Integration
1.6 The student will acknowledge that materials are created by others (e.g., author, illustrator).	English: 1.14c (identify pictures, texts, or people as sources of information)
2.6 The student will acknowledge that materials are created by others (e.g., author, illustrator, and website).	English: 2.12f (describing difference between plagiarism and using own words)
3.7 The student will give credit to sources when borrowing or changing ideas (e.g., using information and pictures created by others, using music created by others, remixing programming projects).	English: 3.10e,f (review writing to check that the language and/or thoughts of another author are given proper credit) Social Studies: 3.1a (primary and secondary sources); 3.1j (accessing variety of media)
4.7 The student will give credit to sources when borrowing or changing ideas (e.g., using information, pictures created by others, using music created by others, remixing programming projects).	English: 4.9d,e (avoiding plagiarism, giving credit)
5.6 The student will give credit to sources when borrowing or changing ideas (e.g., using information, pictures created by others, using music created by others, remixing programming projects).	English: 5.9d,e (avoid plagiarism by giving credit whenever using another person's media, facts, graphics, music, and quotations)

ALPR Module 7. Additional Resources

Here are some additional resources that may be of interest to you. Please note that ODU is not responsible for the content contained on external sites.

***We recommend that you right click on the links and choose "Open in New Tab" for best viewing.**

- **Citing Programming Code - Computer Science & Computer Engineering - Research Guides at University of Arkansas (libguides.com) (https://uark.libguides.com/CSCE/CitingCode)**
- **Licenses & Standards | Open Source Initiative (https://opensource.org/licenses)**
- **Why is citing sources important? - Citing Sources & Style Manuals - Research Guides at Embry-Riddle Aeronautical University (erau.edu) (https://guides.erau.edu/citestyle/importance#:~:text=Citing%20your%20sources%20is%20important%20for%20a%20variety,more%20information%20about%20your%20sources%20It%20prevents%20plagiarism%21)**
- **Why Cite? Three Reasons to Cite Your Sources - Plagiarism Today (https://www.plagiarismtoday.com/2017/05/16/why-cite/)**
- **6 Reasons Why Citation of Sources is Important When Writing (falconediting.com) (https://falconediting.com/en/blog/6-reasons-why-citation-of-sources-is-important-when-writing)**
- **4 Main Reasons Why You Should Cite Your Sources (audience-advantage.com) (https://www.audience-advantage.com/four-reasons-why-you-should-cite-sources/)**

Published | Preview | Assign To | Edit

ALPR Module 7 Assessment

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Quiz Type	Graded Quiz
Points	99
Assignment Group	Imported Assignments
Shuffle Answers	No
Time Limit	No Time Limit
Multiple Attempts	Yes
Score to Keep	Highest
Attempts	Unlimited
View Responses	Always
Show Correct Answers	Immediately
One Question at a Time	No
Require Respondus LockDown Browser	No
Required to View Quiz Results	No

Due	For	Available from	Until
-	Everyone	-	-

ALPR Module 7 Assessment

This is a preview of the published version of the quiz

Started: Sep 19 at 5:33pm

Quiz Instructions

This assessment is designed to test your content knowledge for the Algorithms and Programming microcredential course. You must earn at least 65 percent to receive a passing score but **you can take the test as many times as needed** to earn the needed score.

Question 1 33 pts

Resources found online should be acknowledged as having their own:

- Financial value
- Author/creator
- Website
- Bugs

Question 2 33 pts

Which of the following, when created by others, should be cited when used? Select all that apply.

- [ ] Conditional statements
- [ ] Uncommon information
- [ ] Algorithms
- [ ] Portions of someone's program
- [ ] Images
- [ ] Music

Question 3 33 pts

Why is it important to give credit where it is due? Select all that apply.

- [ ] To make money off commission
- [ ] It is a small token of appreciation
- [ ] It validates your research
- [ ] To acknowledge the creator of the material

Not saved Submit Quiz

Return to Course Materials (Module 7)

Congratulations, you have completed all of the modules for this course!

Click here to return to Course Materials. (https://canvas.odu.edu/courses/185316/pages/algorithms-and-programming-modules)

Click here to go to the Lesson Plan Assignment module. (https://canvas.odu.edu/courses/185316/pages/lesson-planning-instructions-and-resources)

Lesson Planning Instructions and Resources

In addition to completing the Final Assessment for this microcredential, you also need to submit a lesson plan that incorporates the CS SOL standards covered in this course along with a self-reflection statement.

This module contains information designed to assist you in this assignment.

Instructions for the assignment:

1. Select a grade level appropriate Computer Science standard or standards associated with any of the modules contained in this microcredential and design a lesson for your students using the 5E lesson format. The lesson can include computer science standards integrated with a core content area standard or computer science standards only.
2. Use the information contained in the Curricular Alignment and Curriculum Framework sections of the modules as well as the Lesson Plan Organizer, Blooms Taxonomy chart and Lesson Plan Checklist contained in this section to assist you in your lesson design.
3. Design your final lesson on the Lesson Plan Template attached to these instructions and save your lesson plan as Your Name_Lesson Title.
4. Complete your Self-Reflection and save as YourName_Reflection.
5. Upload your lesson plan and self reflection using the "Lesson Plan Submission" link below, or navigate through the module and use the submission link on the final page.

Click here to submit your Lesson Plan Assignment (https://canvas.odu.edu/courses/185316/assignments/2703158)

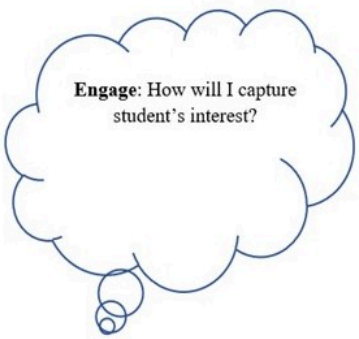
Lesson Planning using the 5E Model

The information contained in the Curricular Alignment and Curriculum Framework sections of each module is based on the Virginia Computer Science SOL's and is designed to provide information that will assist in the development of a Computer Science integrated lesson plan.

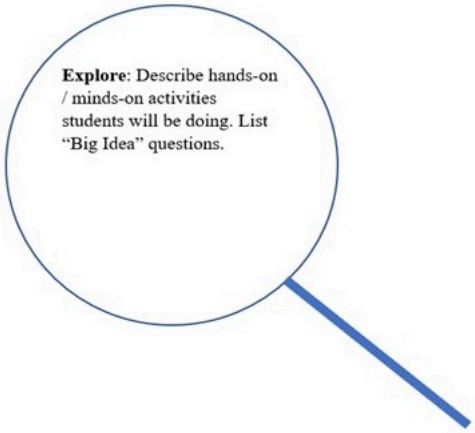
Attached are the 5E Lesson Plan Brainstorm document and the ARCS Lesson Plan template. The Brainstorm document is an organizer that provides a place to take notes as you progress through the information contained in the modules, and you will use the lesson plan template in the development of your final lesson plan for submission. Both Word and PDF versions are attached for your convenience.

Also attached are the ARCS Lesson Plan Checklist, a rubric-like document that allows for self-assessment, and **instructions for the required lesson plan self-reflection.**

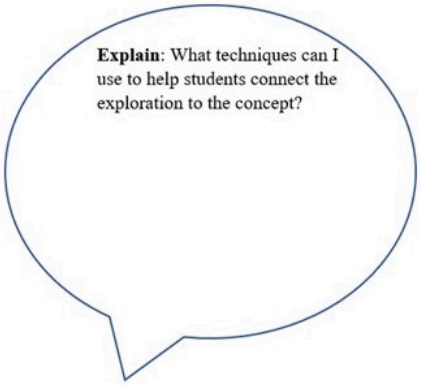
When you identify one or more concepts contained in the Virginia Computer Science Standards for use in a lesson, please use the 5 E lesson organizer to assist you in gathering information for your lesson. The following icons contain an explanation of what should take place during each part of the 5E lesson.



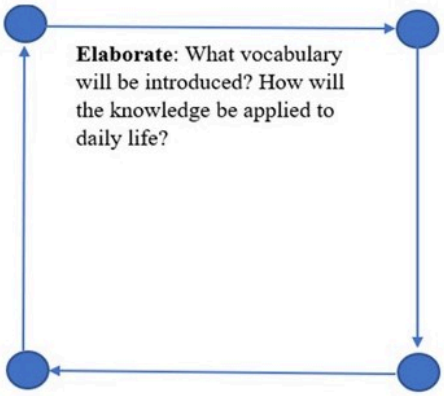
Engage - Prepare the student for the lesson by the use of a thought provoking, engaging short activity or problem. This part of the lesson should make the student curious and excited to learn more about the topic. This may be where prior knowledge is assessed.



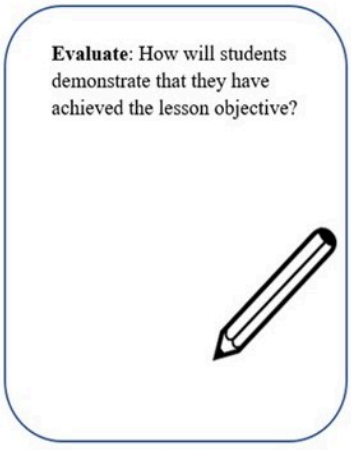
Explore - Students explore the topic through concrete experiences. This is where the student engages in hands-on instructional experiences. This stage of the lesson is student driven and the teacher serves in the role of facilitator.



Explain – This is the section of the lesson where concepts explored are clarified through teacher lead instruction. Students should be able to make connections between their hands-on experiences, that took place during the explore stage, to subject area concepts or theory.



Elaborate - Students demonstrate information learned through new experiences. This part of the lesson is an opportunity to determine misconceptions prior to the evaluation.



Evaluate - Formal or informal assessment. This can include multiple types of assessment including student tests, performance assessments, classwork, and verbal or written responses.

Lesson Plan Brainstorm V2.pdf (https://canvas.odu.edu/courses/185316/files/44845383/download?wrap=1) (https://canvas.odu.edu/courses/185316/files/44845383/download?download_frd=1)

ARCS Microcredential Lesson Plan Self-Reflection Instructions.pdf
(https://canvas.odu.edu/courses/185316/files/44845381/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845381/download?download_frd=1)

ARCS Microcredential Lesson Plan Checklist.pdf
(https://canvas.odu.edu/courses/185316/files/44845390/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845390/download?download_frd=1)

ARCS MC Lesson Plan Template.pdf
(https://canvas.odu.edu/courses/185316/files/44845373/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845373/download?download_frd=1)

ARCS Microcredential Lesson Plan Self-Reflection Instructions.docx
(https://canvas.odu.edu/courses/185316/files/44845644/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845644/download?download_frd=1)

ARCS Microcredential Lesson Plan Checklist.docx
(https://canvas.odu.edu/courses/185316/files/44845649/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845649/download?download_frd=1)

ARCS MC Lesson Plan Template.docx
(https://canvas.odu.edu/courses/185316/files/44845650/download?wrap=1)
(https://canvas.odu.edu/courses/185316/files/44845650/download?download_frd=1)

Lesson Plan Brainstorm V2.docx (https://canvas.odu.edu/courses/185316/files/44845404/download?wrap=1) (https://canvas.odu.edu/courses/185316/files/44845404/download?download_frd=1)

Lesson Plan Brainstorm

Fill in the icons with a lesson component ideas

Elaborate: What vocabulary will be introduced? How will the knowledge be applied to daily life?

Evaluate: How will students demonstrate that they have achieved the lesson objective?

ARCS Microcredential Lesson Plan Self-Reflection Instructions

Please submit a statement of self-reflection for the lesson plan assignment for this course. Your self-reflection should be in narrative format and be no longer than 250 words.

There is no template for this part of the assignment, but please ensure that all of the following are present:

- Reflection offers insight into why the topic(s) and SOL(s) were chosen as the focus of the lesson.
- Reflection describes how the teacher drew on their content knowledge to design the lesson.
- Reflection describes how teacher drew on their pedagogical knowledge to design the lesson.
- Reflection describes how the lesson could be modified to support one or more groups of diverse learners.

ARCS Microcredential Lesson Rubric

Part A. Lesson Plan Format and Instructional Goals

Teacher competency:	**Pass**
The lesson plan follows the 5E lesson format.	All of the following are present: ☐ The lesson is organized in the 5E format. ☐ All of the E sections are present: Engage, Explore, Explain, Elaborate, and Evaluate. ☐ The topic of the lesson is evident in all of the E sections.
The lesson plan includes instructional goals and objectives, sometimes referred to as Learning Targets.	All of the following are present: ☐ Instructional goals/objectives are clearly labeled. ☐ What the learner will know and be able to do are clearly stated. ☐ The Evaluate section addresses students' acquisition of the instructional goals and objectives.

Part B. SOL Content Selection and Integration

Teacher competency:	**Pass**
The lesson is designed around a Virginia Computer Science SOL. A content area SOL is optional except in the Lesson Integration Microcredential.	All of the following are present: ☐ The grade level CS SOL is clearly identified. ☐ Key vocabulary terms are presented. ☐ The Engage or Explore portions of the lesson allow students to enact the CS Standard, Skills and Concepts.
**Lesson Integration Microcredential Lesson Plan: The lesson includes a content area SOL as well as a CS SOL.	All of the following are present: ☐ The grade level content area SOL (e.g. math, science) is clearly identified. ☐ Key vocabulary terms for the content area SOL are presented. ☐ Two or more portions of the lesson plan allow students to enact CS and content area Standards, Skills, and Concepts.

Part C. Instructional Delivery

Teacher competency:	Pass
The lesson describes grade level appropriate instructional strategies.	All of the following are present: ☐ For each of the 5E sections, an appropriate instructional strategy is described. ☐ For each of the 5E sections, the lesson plan includes information about anticipated teacher and student behavior. ☐ The instructional strategies allow for the student to demonstrate the actions listed in the SOL (e.g. create, construct).
The lesson includes appropriate materials and technology.	All of the following are present: ☐ Materials (and technology, if needed) are developmentally appropriate. ☐ All necessary materials (and technology, if needed) are clearly listed. ☐ The selected materials (and technology, if needed) enhance learning.

Part D. Teacher reflection

Teacher competency:	Pass
A statement is included in which the teacher reflects on their own areas of professional growth.	All of the following are present: ☐ Reflection offers insight into why the topic(s) and SOL(s) were chosen as the focus of the lesson. ☐ Reflection describes how the teacher drew on their content knowledge to design the lesson. ☐ Reflection describes how teacher drew on their pedagogical knowledge to design the lesson. ☐ Reflection describes how the lesson could be modified to support one or more groups of diverse learners.

ARCS Lesson Plan Template

Lesson Title:	**Duration:**
CS Standard:	**Content area standard (if applicable):**
Essential Question(s):	
Student Objectives: I can …	
Vocabulary:	
Differentiation strategies:	
Resources:	
Engage:	
Explore:	
Explain:	
Elaborate:	
Evaluate:	

Attachments (as needed)

Student materials