Animate a Joke

Minimum experience: Grades 3+, 1st year using Scratch, 3rd quarter or later

At a Glance

Overview and Purpose

Coders combine understandings from several prior projects to animate a joke. The purpose of this project is to reinforce prior understandings with a focus on modularity.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will review how to animate sprites to make them look like they are talking. **Question:** ● How can we animate sprites to make them look like they are talking?	**Statement:** ● I will create a project with at least ## animated joke(s). *(use a number appropriate for the amount of time and experience levels for the coders you work with)* **Question:** ● How can we create a project with at least ## animated joke(s)? *(use a number appropriate for the amount of time and experience levels for the coders you work with)*
Main standard(s):	**Reinforced standard(s):**
1B-AP-10 Create programs that include sequences, events, loops, and conditionals ● Control structures specify the order (sequence) in which instructions are executed within a program and can be combined to support the creation of more complex programs. Events allow portions of a program to run based on a specific action. For example, students could write a program to explain the water cycle and when a specific component is clicked (event), the program would show information about that part of the water cycle. Conditionals allow for the execution of a portion of code in a program when a certain condition is true. For example, students could write a math game that asks multiplication fact questions and then uses a conditional to check whether or not the answer that was entered is correct. Loops allow for the repetition of a sequence of code multiple times. For example, in a program that produces an animation about a famous historical character, students could use a loop	**1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features. ● Programs can be broken down into smaller parts, which can be incorporated into new or existing programs. For example, students could modify prewritten code from a single-player game to create a two-player game with slightly different rules, remix and add another scene to an animated story, use code to make a ball bounce from another program in a new basketball game, or modify an image created by another student. (source) **1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences. ● Planning is an important part of the iterative process of program development. Students outline key features, time and resource constraints, and user expectations. Students should document the plan as, for example, a storyboard, flowchart, pseudocode, or story map. (source)

to have the character walk across the screen as they introduce themselves. (source)

1B-AP-11 Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.

- Decomposition is the act of breaking down tasks into simpler tasks. For example, students could create an animation by separating a story into different scenes. For each scene, they would select a background, place characters, and program actions. (source)

1B-AP-15 Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.

- As students develop programs they should continuously test those programs to see that they do what was expected and fix (debug), any errors. Students should also be able to successfully debug simple errors in programs created by others. (source)

1B-AP-17 Describe choices made during program development using code comments, presentations, and demonstrations.

- People communicate about their code to help others understand and use their programs. Another purpose of communicating one's design choices is to show an understanding of one's work. These explanations could manifest themselves as in-line code comments for collaborators and assessors, or as part of a summative presentation, such as a code walk-through or coding journal. (source)

Practices and Concepts

Source: K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

Main practice(s):	Reinforced practice(s):
Practice 5: Creating computational artifacts ● "The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps." (p. 80) ● **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80) ● **P5.3.** Modify an existing artifact to improve or customize it. (p. 80)	**Practice 6: Testing and refining computational artifacts** ● "Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts." (p. 81) ● **P6.1.** Systematically test computational artifacts by considering all scenarios and using test cases." (p. 81) ● **P6.2.** Identify and fix errors using a systematic process. (p. 81) **Practice 7: Communicating about computing** ● "Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences." (p. 82) ● **P7.2.** Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. (p. 82)
Main concept(s):	**Reinforced concept(s):**
Control	**Algorithms**

- "Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution." (p. 91)
- **Grade 5 -** "Control structures, including loops, event handlers, and conditionals, are used to specify the flow of execution. Conditionals selectively execute or skip instructions under different conditions." (p. 103)

Modularity

- "Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex. In early grades, students learn that algorithms and programs can be designed by breaking tasks into smaller parts and recombining existing solutions. As they progress, students learn about recognizing patterns to make use of general, reusable solutions for commonly occurring scenarios and clearly describing tasks in ways that are widely usable." (p. 91)
- **Grade 5 -** "Programs can be broken down into smaller parts to facilitate their design, implementation, and review. Programs can also be created by incorporating smaller portions of programs that have already been created." (p. 104)

- "Algorithms are designed to be carried out by both humans and computers. In early grades, students learn about age-appropriate algorithms from the real world. As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms." (p. 91)
- **Grade 5 -** "Different algorithms can achieve the same result. Some algorithms are more appropriate for a specific context than others." (p. 103)

Scratch Blocks	
Primary blocks	Events, Looks, Motion
Supporting blocks	Control, Operators, Sound

Vocabulary	
Algorithm	• A step-by-step process to complete a task. (source) • An algorithm is a formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point. (source)
Backdrop	• One out of possibly many frames, or backgrounds, of the Stage. (source)
Debugging	• The process of finding and correcting errors (bugs) in programs. (source) • To find and remove errors (bugs) from a software program. Bugs occur in programs when a line of code or an instruction conflicts with other elements of the code. (source)
Event (trigger)	• An action or occurrence detected by a program. Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory. Most modern applications, particularly those that run in Macintosh and Windows environments, are said to be event-driven,because they are designed to respond to events. (source) • The computational concept of one thing causing another thing to happen. (source) • Any identifiable occurrence that has significance for system hardware or software. User-generated events include keystrokes and mouse clicks; system-generated events include program loading and errors. (source)

Parallel	• Refers to processes that occur simultaneously. Printers and other devices are said to be either parallel or serial. Parallel means the device is capable of receiving more than one bit at a time (that is, it receives several bits in parallel). Most modern printers are parallel. (source) • The computational concept of making things happen at the same time. (source)
Scripts	• One or more Scratch blocks connected together to form a sequence. Scripts begin with an event block that responds to input (e.g., mouse click, broadcast). When triggered, additional blocks connected to the event block are executed one at a time. (source)
Sprite	• A media object that performs actions on the stage in a Scratch project. (source)
Storyboard	• Like comic strips for a program, storyboards tell a story of what a coding project will do and can be used to plan a project before coding.
More vocabulary words from CSTA	• Click here for more vocabulary words and definitions created by the Computer Science Teachers Association

Connections

Integration	**Potential subjects:** Language arts, media arts, physical education, science **Example(s):** This project could integrate with language arts lessons if coders created jokes with fictional characters. This project could integrate with physical education classes if coders predicted and embodied a sprite's motion by physically mimicking a sprite's algorithm. Note, this process may get a little silly in the best way possible. Click here to see other examples and share your own ideas on our subforum dedicated to integrating projects or click here for a studio with similar projects.
Vocations	Authors, marketers, and media artists are often asked to create a story to sell a product or create a narrative. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

- Example project
- Video walkthroughs
- Quick reference guide
- Project files
- School appropriate jokes ;)

Project Sequence

Preparation (20+ minutes)

Suggested preparation	Resources for learning more

Customizing this project for your class (10+ minutes): Remix the project example to include your own animated jokes.
 	
(10+ minutes) Read through each part of this lesson plan and decide which sections the coders you work with might be interested in and capable of engaging with in the amount of time you have with them. If using projects with sound, individual headphones are very helpful.
	• BootUp Scratch Tips
	  ○ Videos and tips on Scratch from our YouTube channel
	• BootUp Facilitation Tips
	  ○ Videos and tips on facilitating coding classes from our YouTube channel
	• Scratch Starter Cards

Download the offline version of Scratch: Although hopefully infrequent, your class might not be able to access Scratch due to Scratch's servers going down or your school losing internet access. Events like these could completely derail your lesson plans for the day; however, there is an offline version of Scratch that coders could use when Scratch is inaccessible. Click here to download the offline version of Scratch on to each computer a coder uses and click here to learn more by watching a short video.	○ Printable cards with some sample starter code designed for beginners ● ScratchEd ○ A Scratch community designed specifically for educators interested in sharing resources and discussing Scratch in education ● Scratch Help ○ This includes examples of basic projects and resources to get started ● Scratch Videos ○ Introductory videos and tips designed by the makers of Scratch ● Scratch Wiki ○ This wiki includes a variety of explanations and tutorials

Getting Started (6-10+ minutes)

Suggested sequence	Resources, suggestions, and connections
1. Review and demonstration (2+ minutes): Begin by asking coders to talk with a neighbor for 30 seconds about something they learned last time; assess for general understanding of the practices and concepts from the previous project. Explain that today we are going to animate a joke. Display and demonstrate the sample project (or your own remixed version).	**Practices reinforced:** ● Communicating about computing **Video:** Project Preview (1:04) **Video:** Lesson pacing (1:48) This can include a full class demonstration or guided exploration in small groups or individually. For small group and individual explorations, you can use the videos and quick reference guides embedded within this lesson, and focus on facilitating 1-on-1 throughout the process. **Example review discussion questions:** ● What's something new you learned last time you coded? ○ Is there a new block or word you learned? ● What's something you want to know more about? ● What's something you could add or change to your previous project? ● What's something that was easy/difficult about your previous project?
2. Discuss (3+ minutes): Have coders talk with each other about how they might create a project like the one demonstrated. If coders are unsure, and the discussion questions aren't helping, you can model thought processes: "I noticed the sprite moved around, so I think they used a motion block. What motion block(s) might be in the code? What else did you notice?" Another approach might be to wonder out loud by thinking aloud different algorithms and testing them out, next asking coders "what do you wonder about or want to try?" After the discussion, coders will begin working on their project as a class, in small groups, or at their own pace.	**Practices reinforced:** ● Communicating about computing **Note:** Discussions might include full class or small groups, or individual responses to discussion prompts. These discussions which ask coders to predict how a project might work, or think through how to create a project, are important aspects of learning to code. Not only does this process help coders think logically and creatively, but it does so without giving away the answer. **Example discussion questions:**

	● What would we need to know to make something like this in Scratch?
	● What kind of blocks might we use?
	● What else could you add or change in a project like this?
	● What code from our previous projects might we use in a project like this?
	● What kind of jokes might we animate?
	  ● What kind of sprites might we see in that joke?
	  ○ What kind of code might they have?

3. Log in (1-5+ minutes): If not yet comfortable with logging in, review how to log into Scratch and create a new project. If coders continue to have difficulty with logging in, you can create cards with a coder's login information and store it in your desk. This will allow coders to access their account without displaying their login information to others.	**Alternative login suggestion:** Instead of logging in at the start of class, another approach is to wait until the end of class to log in so coders can immediately begin working on coding; however, coders may need a reminder to save before leaving or they will lose their work. **Why the variable length of time?** It depends on comfort with login usernames/passwords and how often coders have signed into Scratch before. Although this process may take longer than desired at the beginning, coders will eventually be able to login within seconds rather than minutes. **What if some coders log in much faster than others?** Set a timer for how long everyone has to log in to their account (e.g., 5 minutes). If anyone logs in faster than the time limit, they can open up previous projects and add to them. Your role during this time is to help out those who are having difficulty logging in. Once the timer goes off, everyone stops their process and prepares for the following chunk.

Project Work (85-90+ minutes; 2+ classes)

Suggested sequence	Resources, suggestions, and connections

4. Create a storyboard (10-15+ minutes):
Walk through the process of creating a storyboard by asking the following questions, then giving coders time to document their answers through physical or digital means:
1. What joke(s) are you going to animate?
2. What might we include in our storyboard for creating an animated joke?
  a. What are some of the media we might use in a project like this (pictures, sounds, sprites, backdrops, etc.)?
3. How will your sprites tell or show the joke?
  a. What code will you use to do that?
4. How might a user interact with the joke?
  a. How might we use code to create that interaction?
 	
When coders are ready, have them show you their storyboard and ask questions for clarification of their intent (which may change once they start coding and get more ideas). If
	Standards reinforced:
	● **1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences
	Practices reinforced:
	● Communicating about computing
	Concepts reinforced:
	● Program development
	● Modularity
	
	Resource: Example storyboard templates
	Resource: Storyboard questions for displaying
	Resource: School appropriate jokes ;)
	
	Note: Some coders do really well with open projects, while others thrive within constraints. It may make more sense to suggest a range of sprites and backdrops so coders aren't overwhelmed with possibilities. This can also help with better predicting how long it might take to create the story.

approved, they may continue on to the next steps (logging in and creating their scenic walk); otherwise they can continue to think through and work on their storyboard.	**Storyboarding Tip:** Coders can color their storyboard (or mark with symbols) what they know, have questions about, and don't know. For example: mark something green if coders know how to create the algorithm for that sprite/action; mark yellow if a coder has questions; mark red if a coder is unsure how to do something. **Suggestion:** If coders need additional help, perhaps pair them with someone who might help them with the storyboarding process. Or, you could have coders meet with a peer to discuss their storyboard before asking to share it with yourself. This can be a great way to get academic feedback and ideas from a peer.
5. Talking sprites (15+ minutes): ***1+ minute demonstration*** Display and demonstrate the sample project (or your own remixed version) one more time. Ask coders to chat with a neighbor about how they might get a sprite to look like it's talking. ***9+ minute reverse engineering*** Ask coders to see if they can figure out how to use the paint editor tools and their code blocks to create an algorithm that makes a sprite do something similar to what was demonstrated. Facilitate by walking around and asking guiding questions. ***5+ minute demonstration*** Walk through each step of the process for creating a talking sprite. Demonstrate one or both methods for moving a mouth or head to simulate talking. Ask coders to image other ways to make a sprite talk (e.g., taking a picture of your mouth in different shapes or expressions). Repeat this process for a couple of sprites and demonstrate how to name the costumes for ease of use. Create a function using message blocks for moving the sprite's mouth by switching to the next and previous costumes to simulate talking (see the video or quick reference guide). Point out that we can use this function multiple times by switching to a desired costume and calling our new function to run in parallel with our other code. To stop our function, we simply need to use stop other scripts in sprite blocks when we want to stop the sprite from talking. Ask how we make sure our sprite doesn't keep their mouth open when we stop our talking function? (use a switch costume to block to switch to a costume with the mouth closed)	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Video:** Talking sprites (6:46) **Quick reference guide:** Click here **Suggested questions:** ● How else could we indicate a sprite is talking in Scratch? ● When should we use text and when should we use recordings to show a talking sprite? ● Why did we use message blocks and not My Blocks? a. (so we can have this function run in parallel with other code every time a sprite talks) **A note on using the "Coder Resources" with your class:** Young coders may need a demonstration (and semi-frequent friendly reminders) for how to navigate a browser with multiple tabs. The reason why is because kids will have at least three tabs open while working on a project: 1) a tab for Scratch, 2) a tab for the Coder Resources walkthrough, and 3) a tab for the video/visual walkthrough for each step in the Coder Resources document. Demonstrate how to navigate between these three tabs and point out that coders will close the video/visual walkthrough once they complete that particular step of a project and open a new tab for the next step or extension. **Although this may seem obvious for many adults, we recommend doing this demonstration the first time kids use the Coder Resources and as friendly reminders when needed**.

6. Animate your joke(s) (60+ minutes, the majority of at least two classes): Give coders time to animate their joke(s) by applying their understandings from previous projects to this new project and encourage them to constantly refer back to their storyboard when they're stuck on what they should do next. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the Tutorials to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions. **Suggested questions:** ● How can you use modularity to make your code more organized and easier to read? ● Can you create hidden sprites a user can interact with in your joke? ● Can you make it so the user helps tell a joke?
7. Add in comments (*the amount of time depends on typing speed and amount of code*): ***1 minute demonstration*** When the project is nearing completion, bring up some code for the project and ask coders to explain to a neighbor how the code is going to work. Review how we can use comments in our program to add in explanations for code, so others can understand how our programs work. Quickly review how to add in comments. ***Commenting time*** Ask coders to add in comments explaining the code throughout their project. Encourage coders to write clear and concise comments, and ask for clarification or elaboration when needed.	**Standards reinforced:** ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing **Concepts reinforced:** ● Algorithms **Video:** Add in comments (1:45) **Quick reference guide:** Click here **Facilitation suggestion:** One way to check for clarity of comments is to have a coder read out loud their comment and ask another coder to recreate their comment using code blocks. This may be a fun challenge for those who type fast while others are completing their comments.

Assessment

Standards reinforced:
- **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations

Practices reinforced:
- Communicating about computing

Although opportunities for assessment in three different forms are embedded throughout each lesson, this page provides resources for assessing both processes and products. If you would like some example questions for assessing this project, see below:

Summative Assessment *of* Learning	**Formative** Assessment *for* Learning	**Ipsative** Assessment *as* Learning
The debugging exercises, commenting on code, and projects themselves can all be forms of summative assessment if a criteria is developed for each project or there are "correct" ways of solving, describing, or creating. **For example, ask the following after a project:** ● Can coders debug the debugging exercises? ● Did coders create a project similar to the project preview? ○ **Note:** The project preview and sample projects are not representative of what all grade levels should seek to emulate. They are meant to generate ideas, but expectations should be scaled to match the experience levels of the coders you are working with. ● Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes? ● Did coders include descriptive comments for each event in all of their sprites? ● Can coders explain how they used broadcast blocks or more blocks as functions to make their code more organized and easier to read (modularity)? ● Can coders explain how their project is similar to their storyboard? ● Did coders create an animated joke with at least ## different jokes? ○ Choose a number appropriate for the coders you work with and the amount of time available.	The 1-on-1 facilitating during each project is a form of formative assessment because the primary role of the facilitator is to ask questions to guide understanding; storyboarding can be another form of formative assessment. **For example, ask the following while coders are working on a project:** ● What are three different ways you could change that sprite's algorithm? ● What happens if we change the order of these blocks? ● What could you add or change to this code and what do you think would happen? ● How might you use code like this in everyday life? ● See the suggested questions throughout the lesson and the assessment examples for more questions.	The reflection and sharing section at the end of each lesson can be a form of ipsative assessment when coders are encouraged to reflect on both current and prior understandings of concepts and practices. **For example, ask the following after a project:** ● How is this project similar or different from previous projects? ● What new code or tools were you able to add to this project that you haven't used before? ● How can you use what you learned today in future projects? ● What questions do you have about coding that you could explore next time? ● See the reflection questions at the end for more suggestions.

Extended Learning

Project Extensions

Suggested extensions	Resources, suggestions, and connections
Use the example project as a guide (as needed) At some point, coders might get stuck or run out of ideas. Rather than explaining to them how to do something, ask them to open the example project, read the comments inside the various sprites and background, and then look at the code to see if they can figure out how to solve their problem. Although this is a very open-ended approach, this models a common coding practice that helps coders become independent learners.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Resource:** Example project **Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the Tutorials to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions. **Note:** The example project has a lot of messages that all do different things. It makes it look like the project is complicated; however, each message/function is relatively easy to understand and has comments that explains what it does.
Add even more (30+ minutes, or at least one class): If time permits and coders are interested in this project, encourage coders to explore what else they can create in Scratch by trying out new blocks and reviewing previous projects to get ideas for this project. When changes are made, encourage them to alter their comments to reflect the changes (either in the moment or at the end of class). While facilitating this process, monitor to make sure coders don't stick with one feature for too long. In particular, coders like to edit their sprites/backgrounds by painting on them or taking photos, or listen to the built-in sounds in Scratch. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the Tutorials to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions. **Suggested questions:** ● What else can you do with Scratch? ● What do you think the other blocks do?

	a. Can you make your project do ____? ● What other sprites can you add to your project? ● What have you learned in other projects that you could use in this project? ● Could you make this project a story or a game with animated sprites? ● What other jokes could you animate in this project?
Similar projects: Have coders explore the code of other peers in their class, or on a project studio dedicated to this project. Encourage coders to ask questions about each other's code. When changes are made, encourage coders to alter their comments to reflect the changes (either in the moment or at the end of class). Watch this video (3:20) if you are unsure how to use a project studio.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features **Practices reinforced:** ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms **Note:** Coders may need a gentle reminder we are looking at other projects to get ideas for our own project, *not to simply play around*. For example, "look for five minutes," "look at no more than five other projects," "find three projects that each do one thing you would like to add to your project," or "find X number of projects that are similar to the project we are creating." **Generic questions:** ● What are some ways you can expand this project beyond what it can already do? ● How is this project similar (or different) to something you worked on today? ● What blocks did they use that you didn't use? a. What do you think those blocks do? ● What's something you like about their project that you could add to your project?
micro:bit extensions: **Note:** the micro:bit requires installation of Scratch Link and a HEX file before it will work with a computer. Watch this video (2:22) and use this guide to learn how to get started with a micro:bit before encouraging coders to use the micro:bit blocks. Much like the generic Scratch Tips folder linked in each Coder Resources document, the micro:bit Tips folder contains video and visual walkthroughs for project extensions applicable to a wide range of projects. Although not required, the micro:bit Tips folder uses numbers to indicate a suggested order for learning about using a micro:bit in Scratch; however, coders who are comfortable with experimentation can skip around to topics relevant to their project.	**Standards reinforced:** ● **1B-AP-09** Create programs that use variables to store and modify data ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process ● **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended **Practices reinforced:** ● Recognizing and defining computational problems ● Creating computational artifacts ● Developing and using abstractions ● Fostering an inclusive computing culture ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control

	• Modularity
	• Program Development
	• Variables
	
	Folder with all micro:bit quick reference guides: Click here
	Additional Resources:
	• Printable micro:bit cards
	  ○ Cards made by micro:bit
	  ○ Cards made by Scratch
	• Micro:bit's Scratch account with example projects
	
	Generic questions:
	• How can you use a micro:bit to add news forms of user interaction?
	• What do the different micro:bit event blocks do and how could you use them in a project?
	• How could you use the LED display for your project?
	• What do the tilt blocks do and how could you use them in your project?
	• How could you use the buttons to add user/player controls?
	• How might you use a micro:bit to make your project more accessible?

Differentiation

Less experienced coders	More experienced coders
Demonstrate the example remix project or your own version, and walk through how to experiment changing various parameters or blocks to see what they do. Give some time for them to change the blocks around. When it appears a coder might need some guidance or has completed an idea, encourage them to add more to the project or begin following the steps for creating the project on their own (or with BootUp resources). Continue to facilitate one-on-one using questioning techniques to encourage tinkering and trying new combinations of code. If you are working with other coders and want to get less experienced coders started with remixing, have those who are interested in remixing a project watch this video (2:42) to learn how to remix a project.	Demonstrate the project without showing the code used to create the project. Challenge coders to figure out how to recreate a similar project without looking at the code of the original project. If coders get stuck reverse engineering, use guiding questions to encourage them to uncover various pieces of the project. Alternatively, if you are unable to work with someone one-on-one at a time of need, they can access the quick reference guides and video walkthroughs above to learn how each part of this project works. If you are working with other coders and want to get more experienced coders started with reverse engineering, have those who are interested watch this video (2:30) to learn how to reverse engineer a project.

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
Why doesn't Jodi appear until after she says supplies? • We need to use a "say" block without a timer, then we need to use a blank "say" block at the end to stop saying the word. • The reason for this is because the "say" block without the timer will immediately	**Standards reinforced:** • **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended **Practices reinforced:** • Testing and refining computational artifacts **Concepts reinforced:** • Algorithms

begin the code beneath it, but the "say" block with the timer will wait until the time is up before moving to the next block (kind of like the "broadcast message" and "broadcast message and wait" blocks).

Why does our code mess up when Champ99 starts asking the joke?

- In our "Champ99 asking" function (message), we "broadcast Champ99 asking" again, which creates a recursive loop (a function that calls itself repeatedly - which can create an infinite loop).
- We are supposed to call our function that makes Champ99 talk: "Champ99 mouth moving."

Why does Champ99 move when Cassy talks, but doesn't move his mouth when he talks?

- In the Champ99 sprite, we need the final function (message) to be "Champ99 mouth moving" and not "Cassy mouth moving."

Even more debugging exercises

- Control

Suggested guiding questions:

- What should have happened but didn't?
- Which sprite(s) do you think the problem is located in?
- What code is working and what code has the bug?
- Can you walk me through the algorithm (steps) and point out where it's not working?
- Are there any blocks missing or out of place?
- How would you code this if you were coding this algorithm from Scratch?
- Another approach would be to read the question out loud and give hints as to what types of blocks (e.g., motion, looks, event, etc.) might be missing.

Reflective questions when solved:

- What was wrong with this code and how did you fix it?
- Is there another way to fix this bug using different code or tools?
- *If this is not the first time they've coded:* How was this exercise similar or different from other times you've debugged code in your own projects or in other exercises?

Unplugged Lessons and Resources

Although each project lesson includes suggestions for the amount of class time to spend on a project, BootUp encourages coding facilitators to supplement our project lessons with resources created by others. In particular, reinforcing a variety of standards, practices, and concepts through the use of unplugged lessons. Unplugged lessons are coding lessons that teach core computational concepts without computers or tablets. You could start a lesson with a short, unplugged lesson relevant to a project, or use unplugged lessons when coders appear to be struggling with a concept or practice.

List of 100+ unplugged lessons and resources

Incorporating unplugged lessons in the middle of a multi-day project situates understandings within an actual project; however, unplugged lessons can occur before or after projects with the same concepts. **An example for incorporating unplugged lessons:**

Lesson 1.	Getting started sequence and beginning project work
Lesson 2.	Continuing project work
Lesson 3.	Debugging exercises and unplugged lesson that reinforces concepts from a project
Lesson 4.	Project extensions and sharing

Reflection and Sharing

Reflection suggestions	Sharing suggestions
Coders can either discuss some of the following prompts with a neighbor, in a small group, as a class, or respond in a physical or digital journal**.** If reflecting in smaller groups or individually, walk around and ask questions to encourage deeper responses and assess for understanding. Here is a sample of a digital	**Standards reinforced:** ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing

journal designed for Scratch (source) and here is an example of a printable journal useful for younger coders.

Sample reflection questions or journal prompts:

- How did you use computational thinking when creating your project?
- What's something we learned while working on this project today?
    - What are you proud of in your project?
    - How did you work through a bug or difficult challenge today?
- What other projects could we do using the same concepts/blocks we used today?
- What's something you had to debug today, and what strategy did you use to debug the error?
- What mistakes did you make and how did you learn from those mistakes?
- How did you help other coders with their projects?
    - What did you learn from other coders today?
- What questions do you have about coding?
    - What was challenging today?
- Why are comments helpful in our projects?
- How is this project similar to other projects you've worked on?
    - How is it different?
- How else could you animate a joke in Scratch?
- What other things could you animate (e.g., newscast, book report, short stories, etc.)
    - What code would you use to create those animations?
- More sample prompts

- Fostering an inclusive culture

Concepts reinforced:

- Algorithms
- Control
- Modularity
- Program development

Peer sharing and learning video: Click here (1:33)
At the end of class, coders can share with each other something they learned today. Encourage coders to ask questions about each other's code or share their journals with each other. When sharing code, encourage coders to discuss something they like about their code as well as a suggestion for something else they might add.

Publicly sharing Scratch projects: If coders would like to publicly share their Scratch projects, they can follow these steps:

1. **Video:** Share your project (2:22)
    a. Quick reference guide
2. **Video (Advanced):** Create a thumbnail (4:17)
    a. Quick reference guide