Pass It On (Story Starters)

Minimum experience: Grades 3+, 1st year using Scratch, 4th quarter or later

At a Glance

Overview and Purpose

Coders engage in a multi-day project where they create or remix an introduction to a short story. The short version of this project is completed by one coder, while the longer version involves coders passing their incomplete project to a peer, who adds to their story. The purpose of this project is to encourage coders to communicate and learn from their peers.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will review coding concepts and practices learned this year to create and remix stories with multiple scenes. **Question:** ● How can we use what we learned this year to create and remix stories with multiple scenes?	**Statement:** ● I will use a storyboard as a guide for creating my own project and adding to projects by my peers. **Question:** ● How can we use a storyboard as a guide for creating our own projects and adding to projects created by peers?
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-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. ● Different algorithms can achieve the same result, though sometimes one algorithm might be most appropriate for a specific situation. Students should be able to look at different ways to solve the same task and decide which would be the best solution. For example, students could use a map and plan multiple algorithms to get from one point to another. They could look at routes suggested by mapping software and change the route to something that would be better, based on which route is shortest or fastest or would avoid a problem. Students might compare algorithms that describe how to get ready for school. Another example might be to write different algorithms to draw a regular polygon and determine which algorithm would be the easiest to modify or repurpose to draw a different polygon. (source) **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.

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

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)
- 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):

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)
- **P5.1.** Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations. (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)

Reinforced practice(s):

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):**
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) **Program Development** ● "Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing." (p. 91) ● **Grade 5 -** "People develop programs using an iterative process involving design, implementation, and review. Design often involves reusing existing code or remixing other programs within a community. People continuously review whether programs work as expected, and they fix, or debug, parts that do not. Repeating these steps enables people to refine and improve programs." (p. 104)	**Algorithms** ● "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) **Control** ● "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)

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

Vocabulary	
Iterative	● Involving the repeating of a process with the aim of approaching a desired goal, target, or result (source) ● Iteration is a single pass through a group of instructions. Most programs contain loops of instructions that are executed over and over again. The computer iterates through the loop, which means that it repeatedly executes the loop. (source) ● The computational practice of developing a little bit, then trying it out, then developing some more. (source)

Modularity	• The characteristic of a software/web application that has been divided (decomposed) into smaller modules. An application might have several procedures that are called from inside its main procedure. Existing procedures could be reused by recombining them in a new application (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)
Remix	• The process of creating something new from something old. Originally a process that involved music, remixing involves creating a new version of a program by recombining and modifying parts of existing programs, and often adding new pieces, to form new solutions. (source) • A creative work that is derived from an original work (or from another remix). A remix typically introduces new content or stylistic elements, while retaining a degree of similarity to the original work. (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:** Any **Example(s):** Rather than using the example story starters, you could include starter prompts or projects related to any subject. This would allow this project to connect with any subject area, topic, or theme. 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. These products often undergo iterative cycles or are passed on to other team members, which is reinforced through this project. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

- Scratch studio with project starters
- Video walkthroughs
- Quick reference guides
- Project files
- This project was inspired by a project in the Creative Computing Guide

Project Sequence

Preparation (20+ minutes)

Suggested preparation	Resources for learning more

Customizing the project starters for your class (10+ minutes per project starter): Remix one of the projects in this studio to create your own project starter.
 	
(10+ minutes) Read through each part of this lesson plan ***(there is a short and long version of this project)*** 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.
 	
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.
	• 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
	  ○ 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 (13-22+ 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. ***If doing the shorter project:*** Explain that today we are going to pick a story starter and remix it to add an ending. ***If doing the longer project:*** Explain that today we are going to start a project that will last for multiple classes. We will begin today by adding on to a starter project, then spend the next couple classes adding to other people's projects, who will also add new sections to your own story each day. On the final day we will return to our original project and see what others have added to our original project.	**Practices reinforced:** • Communicating about computing **Video:** Project Preview (4:36) **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, it might help to set a time limit for exploration before discussing the project. **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. Log in (1-5+ minutes): If not yet comfortable with logging in, review how to log into Scratch. If coders continue to have difficulty with logging in, you can create cards with a coder's login information and store it in	**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.

your desk. This will allow coders to access their account without displaying their login information to others.	**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.
3. Explore and discuss (10-15+ minutes): Once they log in, have them navigate to this studio (or your own) and explore project starters that look interesting. Encourage coders to think through what would happen next in the story before exploring another project starter. Bring up one of the project starters from this studio and have coders talk with each other about what they would add next to this story. Use some of the example discussion questions to guide a discussion on a couple of the project starters.	**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 do you think could happen next in the story? ○ What background would we use for the next scene? ○ What sprites would we see on that scene? ○ What would happen to them? ● When would the next scene end?

(Shorter) Project Work (45-50+ minutes; 1+ classes)

Suggested sequence	Resources, suggestions, and connections
3. Create a storyboard for the rest of the story (15-20+ minutes): Either hand out paper, use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create for the conclusion of their chosen project starter from the project starter studio (or your own remixed version). It may help to model this process with a separate set of random ideas. Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them. 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 approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on	**Standards reinforced:** ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. ● **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 **Suggested storyboard questions:** ● Which project starter are you selecting? ● What will happen next in the story?

their storyboard.
 	
Note: Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas.
	• What backdrop will you use for the next scene?
	  ○ What sprites are going to be in that scene?
	  ○ What will the sprites do?
	  ○ What kind of code might we use to do that?
	• How will the story end?
	• What are all of the ways we can interact with the story?
	  ○ In each of these ways we can interact with the story, how might we use code to create that interaction?
	
	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.

4. Finish the story (25+ minutes, or the majority of the class): Ask coders to finish their story starter from this studio using their storyboard. Facilitate by walking around and asking questions and encouraging coders to try out new blocks. Remind everyone they can look at the original code, but they can only add and change code for the section of the story they are adding. Before moving to the final section of this project, ask coders to add comments in their project (see the resources on the right) and share their project in a class studio dedicated to this project.	**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. • **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** • Communicating about computing • Creating computational artifacts • Testing and refining computational artifacts **Concepts reinforced:** • Algorithms • Control • Modularity **Video:** Add in comments (1:45) **Quick reference guide:** Click here **Suggested questions:** • What sounds might we hear in this story? • How could we add even more to our story than what's in our storyboard? • How will this story build off the story starter? **A note on using the "Coder Resources" with your class:** Young codes 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**.
5. Share your story (5+ minutes) Ask coders to share their story with someone who worked on the same story starter or a story starter that looked interesting to them. Encourage discussion around what they like about the original and newly added code, something they learned by analyzing the original code, and what they would have done differently if they had created the entire story on their own. Have coders use some of the reflection prompts to reflect on this project.	**Standards reinforced:** ● **1B-AP-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. **Practices reinforced:** ● Communicating about computing **Concepts reinforced:** ● Algorithms ● Control ● Modularity

(Longer) Project Work (120-135+ minutes; 3+ classes)

Suggested sequence	Resources, suggestions, and connections
3. Create a storyboard for the second scene (5-10+ minutes): Either hand out paper, use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create for the second scene of their chosen project starter from the project starter studio (or your own remixed version). It may help to model this process with a separate set of random ideas. Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them. 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 approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on their storyboard. **Note:** Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas.	**Standards reinforced:** ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. ● **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 **Suggested storyboard questions:** ● Which project starter are you selecting? ● What will happen next in the story? ● What backdrop will you use for the next scene? ○ What sprites are going to be in that scene? ○ What will the sprites do? ○ What kind of code might we use to do that? ● When will that scene end? (*note, the scene is ending, not the story*) ● What are all of the ways we can interact with the story? ○ In each of these ways we can interact with the story, how might we use code to create that interaction? **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.

4. Create the second scene of the story (25+ minutes, or the remainder of the first class): For the remainder of class, coders will create the second scene of their story using their storyboard. Facilitate by walking around and asking questions and encouraging coders to try out new blocks and creating functions to make it easy for people to understand each section of code. Near the end of this section or class, ask coders to add comments in their project (see the resources on the right) and share their project in a class studio dedicated to this project.	**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. ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing ● Creating computational artifacts ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Video:** Add in comments (1:45) **Quick reference guide:** Click here **Suggested questions:** ● What sounds might we hear in this scene? ● How could we add even more to our story than what's in our storyboard? ● What do you think might happen in the next scene? **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**.
5. Create a storyboard for the third scene of someone else's story (5-10+ minutes): Either assign each coder to a project or given them a few minutes to virtually swap projects with a friend, who is going to add a new scene to their remixed story by remixing the project from the studio. **Make it clear they are not going to change anyone's code, but are going to add a new scene to a remixed version of this project.** Either hand out paper, use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create for the third scene. It may help to model this process with a separate set of random ideas.	**Standards reinforced:** ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process. ● **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

Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them. 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 approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on their storyboard. **Note:** Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas.	**Resource:** Example storyboard templates **Suggested storyboard questions:** ● What will happen next in the story? ● What backdrop will you use for the next scene? ○ What sprites are going to be in that scene? ○ What will the sprites do? ○ What kind of code might we use to do that? ● When will that scene end? (*note, the scene is ending, not the story*) ● What are all of the ways we can interact with the story? ○ In each of these ways we can interact with the story, how might we use code to create that interaction? **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.
6. Create the third scene of someone else's story (25+ minutes, or the remainder of the second class): Ask coders to create the third scene of a story using their storyboard. Facilitate by walking around and asking questions and encouraging coders to try out new blocks. **Remind everyone they can look at the original code, but they can only add and change code for the third scene.** Near the end of this section or class, ask coders to add comments in their project (see the resources on the right) and share their project in a class studio dedicated to this project.	**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. ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing ● Creating computational artifacts ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Video:** Add in comments (1:45) **Quick reference guide:** Click here **Suggested questions:** ● What sounds might we hear in this scene? ● How could we add even more to our story than what's in our storyboard? ● How will this scene build off the previous scenes? ● What do you think might happen in the next scene?
7. Create a storyboard for the final scene of someone else's story (5-10+ minutes): Either assign each coder to a project or given them a few minutes to virtually swap projects with a friend, who is going	**Standards reinforced:** ● **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.

to add the final scene to their recently remixed story by remixing the project from the studio. **Make it clear they are not going to change anyone's code, but are going to add a final scene to a remixed version of this project.**
 	
Either hand out paper, use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create for the final scene. It may help to model this process with a separate set of random ideas.
 	
Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them.
 	
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 approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on their storyboard.
 	
Note: Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas.
	• **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
	
	Suggested storyboard questions:
	• What will happen at the end of this story?
	• What backdrop will you use for the final scene?
	  ○ What sprites are going to be in that scene?
	  ○ What will the sprites do?
	  ○ What kind of code might we use to do that?
	• How will the story end?
	• What are all of the ways we can interact with the story?
	  ○ In each of these ways we can interact with the story, how might we use code to create that interaction?
	
	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.

8. Create the final scene of someone else's story (25+ minutes, or the majority of the third class): Ask coders to create the final scene of a story using their storyboard. Facilitate by walking around and asking questions and encouraging coders to try out new blocks. **Remind everyone they can look at the original code, but they can only add and change code for the final scene.**	**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. • **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** • Communicating about computing • Creating computational artifacts • Testing and refining computational artifacts **Concepts reinforced:** • Algorithms • Control • Modularity **Video:** Add in comments (1:45) **Quick reference guide:** Click here **Suggested questions:** • What sounds might we hear in this scene?

	• How could we add even more to our story than what's in our storyboard? • How will this scene build off the previous scenes?
9. Go back to your original story (30+ minutes, or an entire class) ***5+ minutes*** Have coders revisit the story they originally shared on the class studio, open the remixes tab from the project page and navigate to the final remix (see this video or guide for reference), then watch their story. ***10+ minutes*** After watching the story, give them time to explore the code in the final remix project and read the comments for each section. Facilitate by encouraging coders to think through why each algorithm was coded the way it was and to think through other ways to get the same (or similar) results. ***15+ minutes*** Ask coders to share the story with their neighbors and discuss what they learned by analyzing the code in the story. Encourage discussion around what they like about the code, something they learned by analyzing the code, and what they would have done differently if they had created the entire story on their own. Have coders use some of the reflection prompts to reflect on this project.	**Standards reinforced:** • **1B-AP-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. **Practices reinforced:** • Communicating about computing **Concepts reinforced:** • Algorithms • Control • Modularity **Video:** How to find the final remix (1:49) **Quick reference guide:** Click here

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	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:**	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

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 My Blocks as functions to make their code more organized and easier to read (modularity)? ● Can coders explain how each of the scenes they worked on are similar to their storyboards? ● Did coders create a scene that added to the story created by someone else? ○ Choose a number appropriate for the coders you work with and the amount of time available. ● Did coders create at least ## scenes in their projects? ○ Choose a number appropriate for the coders you work with and the amount of time available.	● 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.	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**
Create a project starter (one full class):	**Standards reinforced:**

Either hand out paper, use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create for the first scene of a starter project that will be used with other classes. Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them. 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 approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on their storyboard. **Coders will spend the remainder of the class creating their project starter, which can be shared on a class studio for other coders to remix.**	● **1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes **Practices reinforced:** ● Creating computational artifacts **Concept reinforced:** ● Program development **Resource:** Example storyboard templates **Suggested storyboard questions:** ● What will happen in the project starter? ● What backdrop will you use? ○ What sprites are going to be in the first scene? ○ What will the sprites do? ○ What kind of code might we use to do that? ● When will the scene end? ● What are all of the ways we can interact with the story? ○ In each of these ways we can interact with the story, how might we use code to create that interaction? **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.
Advanced reverse engineering even more ideas (20+ minutes each): ***15+ minute reverse engineering and peer-to-peer coaching*** Ask coders to try and reverse engineer one of the sprites from a starter project without looking at the code. Encourage peer-to-peer coaching as you also facilitate by walking around and asking guiding questions (they may need to practice this). If coders figure out how to get their sprite to do something similar, have them document in their journal, share with a partner, or attempt reverse engineering another sprite in the same or different project starter. ***5+ minute peer explanation and demonstration*** Ask coders to find a partner who is currently working on their project starter as their main project and have them reveal the code, walk through each step of the algorithm, and explain any new blocks. Encourage coders to ask each other questions about the code.	**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 ● **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended **Practices reinforced:** ● Communicating about computing ● Creating computational artifacts ● Fostering an inclusive computing culture ● Testing and refining computational artifact **Concepts reinforced:** ● Algorithms ● Control **Video:** Suggestions for reverse engineering (4:25) **Alternative suggestion:** If reverse engineering is too difficult for the coders you work with, you could display the source code and have coders predict what will happen. **Suggested guiding questions:** ● What kind of blocks do you think you might need to do something like that? ● Do you see a pattern where we might use a repeat? ○ If so, what kind of repeat?

	• When combining ideas, what happens if you use multiple event blocks in the same sprite? *(Note: this concept is referred to as parallel computing)* ○ What are the benefits/drawbacks of using multiple event blocks? **Potential discussion:** There is not always one way to recreate something with code, so coders may come up with alternative solutions to your own code. When this occurs, it can open up an interesting discussion or journal reflection on the affordances and constraints of such code. **Suggested application and exploration questions:** • What other code blocks could you use? • What other sprites might use similar code? • What other code could we add to this project?
Modify the original code (30+ minutes, or at least one class): If time permits and coders are interested in this project, encourage coders to go into their final project and alter the original code that other people created by adding or changing the code to do something even more interesting. When changes are made, encourage coders to test their refinements and alter their comments to reflect the changes (either in the moment or at the end of class).	**Standards reinforced:** • **1B-AP-10** Create programs that include sequences, events, loops, and conditionals **Practices reinforced:** • Creating computational artifacts • Testing and refining 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 can you change in the original code to make this project even more interesting? • How could you simplify or clean up the original code? • What other sights, sounds, or interactions could you add to make the project even more interesting?
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
If coders struggle with this kind of project, pair them with other coders with more experience or understanding. Just make sure the lesser experienced coder "drives" the mouse and the more experienced coder can "navigate." It might also help less experienced coders if they have time to see what others are creating for their stories; encourage coders to walk around and see what others are doing and then adding similar code in their projects.	Challenge coders to figure out how to recreate a project starter 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 click "see inside" and read the comments in the code to learn how each part of the project starter 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 does Captain Monet's mouth move after talking but not when she is talking? ● We need to use the "say" block that doesn't have "for _ seconds" before calling our "Move mouth" function Why doesn't Aiden ask the question about how humans went extinct? ● We need to use a "when I receive Ask a question" event and not a "when backdrop switches to Castle 3" block Why doesn't the Rocketship sprite start at the top of the screen and land in the middle of the screen? ● We need to add a "go to" block with the y coordinate higher than the top of the screen Why doesn't the Referee sprite count down before saying "Sport!" at the end? ● We need to use "say _ for _ seconds" blocks or the code will run so fast we won't see those three blocks Why does this conversation seem out of order? ● The "broadcast message" blocks need to be swapped so it's "Kai responds" and then "Put the collar on" Why don't we ever hear the creepy sound that woke Boo up?	**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 ● 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?

- We need to use the "Setup" and Creepy sound" blocks before our first "say" block. This will reset the sprite and then play the sound.
    - Note, the "wait 1 seconds" block is optional

Why does Champ99 only say one thing and then move his mouth forever?

- We need to use a "repat #" block instead of a "forever" block and put the "say _" outside of the repeat

Why does the Zebra's drinking look so weird?

- We need to call both functions, not just the "Bend down and wait" function

Even more debugging 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 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?	**Standards reinforced:** ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing ● Fostering an inclusive culture **Concepts reinforced:** ● Algorithms ● Control ● Modularity ● Program development

- 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?
- What happened in your story that you didn't expect?
- What code did other people use that was interesting to you?
    - What did you learn by looking at other people's code?
- In what ways might users interact with a story?
- More sample prompts

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