



Tag, You're It!

Minimum experience: Grades 1+, 2nd year using ScratchJr, 1st quarter or later

At a Glance

Overview and Purpose

Coders use a variety of blocks and sprites to create a multi-page story with a moral. The purpose of this project is to apply previously learned concepts in a new context. Although the example project is about lessons learned while playing tag, this project can be adapted to any story with a moral.

Objectives and Standards

Process objective(s):

Statement:

- I will review how to use the ["go to page" blocks](#) to create a story with multiple pages.

Question:

- How can we use the ["go to page" blocks](#) to create a story with multiple pages?

Main standard(s):

1A-AP-10 Develop programs with sequences and simple loops, to express ideas or address a problem.

- Programming is used as a tool to create products that reflect a wide range of interests. Control structures specify the order in which instructions are executed within a program. Sequences are the order of instructions in a program. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. If the commands to program a robot are not in the correct order, the robot will not complete the task desired. Loops allow for the repetition of a sequence of code multiple times. For example, in a program to show the life cycle of a butterfly, a loop could be combined with move commands to allow continual but controlled movement of the character. ([source](#))

1A-AP-12 Develop plans that describe a program's sequence of events, goals, and expected outcomes.

- Creating a plan for what a program will do clarifies the steps that will be needed to create a program and can be used to check if a program is correct. Students could create a planning document, such as a story

Product objective(s):

Statement:

- I will storyboard and create a multi-page story with a moral.

Question:

- How can we storyboard and create a multi-page story with a moral?

Reinforced standard(s):

1A-AP-08 Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks.

- Composition is the combination of smaller tasks into more complex tasks. Students could create and follow algorithms for making simple foods, brushing their teeth, getting ready for school, participating in clean-up time. ([source](#))

1A-AP-11 Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.

- Decomposition is the act of breaking down tasks into simpler tasks. Students could break down the steps needed to make a peanut butter and jelly sandwich, to brush their teeth, to draw a shape, to move a character across the screen, or to solve a level of a coding app. ([source](#))

1A-AP-14 Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops.

- Algorithms or programs may not always work correctly. Students should be able to use various strategies, such as changing the sequence of the steps, following the algorithm in a step-by-step manner, or trial and error to fix problems in algorithms and programs. ([source](#))

map, a storyboard, or a sequential graphic organizer, to illustrate what their program will do. Students at this stage may complete the planning process with help from their teachers. ([source](#))

1A-AP-15 Using correct terminology, describe steps taken and choices made during the iterative process of program development.

- At this stage, students should be able to talk or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. This could be done using coding journals, discussions with a teacher, class presentations, or blogs. ([source](#))

Practices and Concepts

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

Main practice(s):

Practice 4: Developing and Using Abstractions

- "Abstractions are formed by identifying patterns and extracting common features from specific examples to create generalizations. Using generalized solutions and parts of solutions designed for broad reuse simplifies the development process by managing complexity." ([p. 78](#))
- **P4.4.** Model phenomena and processes and simulate systems to understand and evaluate potential outcomes. ([p. 79](#))

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](#))

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):

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

Reinforced concept(s):

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

understanding to combinations of structures that support complex execution." ([p. 91](#))

- **Grade 2** - "Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions." ([p. 96](#))

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 2** - "People develop programs collaboratively and for a purpose, such as expressing ideas or addressing problems." ([p. 97](#))

algorithms, as well as the evaluation of competing algorithms." ([p. 91](#))

- **Grade 2** - People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow." ([p. 96](#))

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 2** - "Complex tasks can be broken down into simpler instructions, some of which can be broken down even further. Likewise, instructions can be combined to accomplish complex tasks." ([p. 97](#))

ScratchJr Blocks

Primary blocks

[Control](#), [Looks](#), [Motion](#), [Triggering](#)

Supporting blocks

[End](#), [Sound](#)

Vocabulary

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](#))

Moral

- The lesson about right and wrong learned from a story or event. ([source](#))
- A warning or lesson we learn from a story

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](#))

Parameter

- A specific piece of information used as input each time a process is run; for example, when a person presses a vending machine button for an item, the information for the item location is sent as a parameter so the machine knows what food to dispense. (Thank you, Peter Rich, for the great example)
- A special kind of variable used in a procedure to refer to one of the pieces of data received as input by the procedure. ([source](#))
- In programming, the term parameter is synonymous with argument, a value that is passed to a routine. ([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: History, language arts, media arts, social studies Example(s): Rather than creating a new story with a moral, coders could recreate a historical, contemporary, or fictional story with a moral, which would allow this project to integrate with history, social studies, or language arts topics.
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

- [Project files](#)
 - Video:** [Downloading project files](#) (1:04)
- [Sample project images](#)

Project Sequence

Preparation (20+ minutes)

Suggested preparation	Resources for learning more
Ensure all devices are plugged in for charging over night. Customizing this project for your class (10+ minutes): Remix the project example to include your own multi-page story with a moral. (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 ScratchJr Tips - Videos and tips on ScratchJr from our YouTube channel BootUp Facilitation Tips - Videos and tips on facilitating coding classes from our YouTube channel Block Descriptions - A document that describes each of the blocks used in ScratchJr Interface Guide - A reference guide that introduces the ScratchJr interface Paint Editor Guide - A reference guide that introduces features in the paint editor Tips and Hints - Learn even more tips and hints by the creators of the app Coding as another language (CAL) - A set of curriculum units for K-2 using both ScratchJr and KIBO robotics ScratchJr in Scratch - If you're using ScratchJr in Scratch, this playlist provides helpful tips and resources

Getting Started (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 create a multi-page story with a moral; a warning or lesson we learn from the story. Provide some examples of stories with a moral. You could read a story, summarize a short story, compare multiple stories, or use the sample project (or your own remixed version). A note on say blocks: If you are displaying a sample project with say blocks, it might help to read the text out loud using various voices for each sprite as it is displayed. This strategy might help early/pre-readers, as well as young coders who are new to learning English. In addition, when young coders begin working on their own project, you can encourage them to use speech-to-text or emojis in their own say blocks instead of typing out words (or use recorded sound blocks).	Practices reinforced: - Communicating about computing Video: Project Preview (1:17) 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. Discuss (8+ 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?” Have coders talk with each other about how they might create a project with a moral. If coders are unsure, brainstorm different stories with morals in them and have the class group up to pick one of the stories to discuss. 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 a multi-page story with a moral? - 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 sprites might we see in a story with a moral? - What kind of code might they have?

Project Work (60+ minutes; 2+ classes)

Suggested sequence	Resources, suggestions, and connections
3. Create a storyboard (15+ minutes): Either hand out paper with at least four different quadrants	Standards reinforced: 1A-AP-12 Develop plans that describe a program’s

(one for each page in ScratchJr), use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create. 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.

sequence of events, goals, and expected outcomes

Practices reinforced:

- Creating computational artifacts

Concept reinforced:

- Program development

Resource: [Example storyboard templates](#)

Suggested storyboard questions:

- What's the moral of the story?
 - What happens first in your story?
 - What happens next?
 - What happens last?
 - What will people learn from your story?
- How many pages will you have?
 - What backdrop will you use for each page?
 - What sprites will we see on each page?
 - When will you go to the next page?
- What will the sprites do on each page?
 - What kind of code might we use to do that?
- What are all of the ways we can interact with the story?
 - Which pages will have user interaction?
 - 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.

Note: This process may take significantly longer than 15 minutes if storyboarding many different sprites in each page. In general, it is best to keep things simple when first creating a project, then adding more complexity if time permits.

4. Create a multi-page story with a moral (45+ minutes):

Ask coders to create a project based on the storyboards they created. Facilitate by walking around and asking questions and encouraging coders to try out new blocks.

Note: The more pages, the more time you can spend on each project. This project could take several days.

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem

Practices reinforced:

- Testing and refining computational artifacts
- Creating computational artifacts
- Developing and using abstractions

Concepts reinforced:

- Algorithms
- Control

Suggested questions:

- What sounds might we hear in each backdrop?
- What do you like about your project?
 - a. What do you want to change?

- How could we add even more to your story than what's in your storyboard?

Assessment

Standards reinforced:

- **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development

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 use a variety of block types in their algorithms and can they explain how they work together for specific purposes? • Can coders explain how their project is similar to their storyboard? • Did coders create a multi-page story with a moral and at least ## different sprites with different algorithms? ○ Choose a number appropriate for the coders you work with and the amount of time available. • Did coders use at least ## pages in their project? ○ Choose a number appropriate for the coders you work with and the amount of time available. ○ Can coders explain when/how the project will switch pages?	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

Switch costumes (15+ minutes)

5+ demonstration

Demonstrate how to use [show and hide blocks](#) with [message blocks](#) to make it so one sprite hides in the same location as another sprite that shows to make it appear as though the sprite is changing costumes. Emphasize the importance of making sure the sprites are in the same location. For example, making them hide into a doorway, then the new sprite (costume) shows in the same location to make it appear as though the costume changed when the came back out of the door. See the [video](#) for a demonstration.

Another option demonstrated in the [video](#) involves changing costumes by switching pages to the same backdrop. Although this option works, you have to make sure every sprite is perfectly lined up to do the transition (which may require some iteration to get the timing correct if the sprite is moving). Each method can work really well depending on the project and purpose.

10+ minute application and exploration

Encourage coders to try and experiment with using one of the two methods for making a sprite appear to change costumes (it doesn't have to relate to this project's theme). Facilitate by walking around and asking questions about their processes.

Creating a transition overlay (Advanced) (15+ minutes)

5+ minute intro demonstration

Demonstrate how other projects (e.g., [A Day at the Beach](#) and [The Story of the Stunky Robot](#)) use a transition overlay in the project. Use the [video](#) to learn the following process:

1. Demonstrate how to create a blank sprite that is all one color.
2. Next, demonstrate creating another blank sprite of the same color, but add your text on one side of the sprite (e.g., something like "The End" or "2 Hours Later").
3. Use [grow blocks](#) to make the sprites as large as possible.

Resources, suggestions, and connections

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem
- **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.

Practices reinforced:

- Testing and refining computational artifacts
- Creating computational artifacts

Concepts reinforced:

- Algorithms
- Control

Video: [Switching costumes](#) (3:20)

Sample questions:

- When might you switch costumes in your projects?
 - What kind of blocks might we use when switching costumes?
 - How can we hide the costume change?
 - For example, moving off the page, hiding behind another sprite, appearing to hide in a door or container, etc.

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem
- **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.

Practices reinforced:

- Testing and refining computational artifacts
- Creating computational artifacts

Concepts reinforced:

- Algorithms
- Control

Video: [Creating a transition overlay](#) (3:38)

Resource: [Paint Editor Guide](#)

4. Drag the blank sprite over to one side of the screen, then drag the sprite with the text on the other side of the screen (the text should now appear centered).
5. Use [show and hide blocks](#) in an algorithm attached to either the [start on green flag or start on message blocks](#) to make both overlays show or hide.

10+ minute application and 1-on-1 facilitating

Quickly review each of the steps for creating a transition overlay (1 - create a blank sprite; 2 - create a sprite with text; 3 - make the sprites big; 4 - position the sprites on the screen; and 5 - trigger the transitions with code). Leave your example code on the screen and give coders time to replicate the process for creating a transition overlay. Facilitate 1-on-1 as needed, but encourage coders to work together.

Facilitation suggestion: This process might be a little complicated for younger coders; however, you can group coders together in small groups to collaboratively create their own transition screen. If coders complete their stinky robot early, encourage them to begin working on their storyboard.

Advanced reverse engineering even more ideas (15+ minutes each page):

1 minute intro demonstration

Demonstrate one of the following example pages on the board without displaying the code:

Page 1

- [Cat](#) and [Tic](#)

Page 2

- [Cat](#) and [Tic](#)

Page 3

- [Cat](#), [Librarian](#), and [Tic](#)

Page 4

- [Cat](#), [Stool](#), [Teacher](#), and [Tic](#)

9+ minute reverse engineering and peer-to-peer coaching

Ask coders to see if they can figure out how to use their code blocks to create algorithms that recreates the entire page. Facilitate by walking around and asking guiding questions.

1 minute explanation demonstration

If coders figured out how to get their sprites to do something similar, have them document in their journal, share with a partner, or have a volunteer show the class their code and thought processes that led to the code. Otherwise, reveal the code, walk through each step of the algorithm, and explain any new blocks.

4+ minute application and exploration

Encourage coders to try something similar, and leave your code up on display while they work. Facilitate by walking around and asking questions

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem
- **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions.
- **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops.

Practices reinforced:

- Communicating about computing
- Testing and refining computational artifacts
- Creating computational artifacts

Concepts reinforced:

- Algorithms
- Control

Video: [Suggestions for reverse engineering](#) (4:25)

Helpful tip: You could add code to loop the page coders are trying to reverse engineer. To do this, add code on the next page that immediately switches back to this page (i.e., on the next page attach a [go to page block](#) to a [start on green flag block](#) to cycle back to the previous page).

Note: It is not recommended to show more than one page at a time, but to show one page, give time for application and exploration, show another page, give time for application and exploration, etc. This process could take multiple classes. Also, some of these examples may be difficult for young coders, so go slow and encourage copying and modifying code as it's good practice.

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:

about how coders might change their code so it's not the same as yours.

- 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?
- What [trigger blocks](#) do you think I used for that sprite?
- Did I use one [trigger block](#) or more than one?
 - What makes you think that?

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?

Adding even more (5+ minutes):

If time permits, encourage coders to explore what else they can create in ScratchJr. Although future lessons will explore different features and blocks, early experimentation should be encouraged.

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. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem

Practices reinforced:

- Testing and refining computational artifacts
- Creating computational artifacts

Concepts reinforced:

- Algorithms
- Control

Suggested questions:

- What else can you do with ScratchJr?
- What do you think the other blocks do?
 - a. Can you make your sprites do ____?
- What other sprites might we use in a story with a moral?
- What other sounds might we hear in our story?
- Can you customize how your sprites look?

Similar projects:

Have coders explore the [sample projects](#) built into ScratchJr (or projects from other coders), and ask them to find code similar to what they worked on today.

Standards reinforced:

- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem

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," or "find three projects that each do one thing you would like to add to your project."

Generic questions:

- 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?

- How might we change the backdrop of this project?
- What other sound or looks blocks might we use in this project?
- Can you turn this project into a story with a moral?

Differentiation

Less experienced coders

ScratchJr is simple enough that it can be picked up relatively quickly by less experienced coders. However, for those who need additional assistance, pair them with another coder who feels comfortable working cooperatively on a project. Once coders appear to get the hang of using ScratchJr, they can begin to work independently.

More experienced coders

Because ScratchJr is not inherently difficult, experienced coders might get bored with simple projects. To help prevent boredom, ask if they would like to be a “peer helper” and have them help out their peers when they have a question. If someone asks for your help, guide them to a peer helper in order to encourage collaborative learning.

Another approach is to encourage experienced coders to experiment with their code or give them an individual challenge or quest to complete within a timeframe.

Debugging Exercises (1-5+ minutes each)

Debugging exercises

Why isn't Tic as fast as the cat?

- [We need to set the speed to fast instead of slow](#)

Why don't we see the cat run across the stage?

- [We need to show the sprite after waiting for 10](#)

Why doesn't the librarian speak when the cat and Tic are running across the screen?

- [We need to use the “start on green flag” block instead of a “start on tap” block](#)

Why does the stool move across the screen multiple times?

- [We need to change the parameter of the move block to move a small amount \(e.g., 4\) instead of a large amount \(e.g., 40\)](#)

[ScratchJr Debugging List](#)

Resources and suggestions

Standards reinforced:

- **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops

Practices reinforced:

- Testing and refining computational artifacts

Concepts reinforced:

- Algorithms
- Control

Display one of the debugging exercises and ask the class what they think we need to fix in our code to get our project to work correctly. Think out loud what might be wrong (e.g., did I use the wrong [trigger block](#), did I forget to repeat something, did I put a block in the wrong place, am I missing blocks, etc.). Ask the class to talk with a neighbor how we might fix the code. Have a volunteer come up to try and debug the code (or demonstrate how). Repeat with each debugging exercise.

Unplugged Lessons and Resources

Standards reinforced:

- **1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks

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](#)

Reflection and Sharing

Reflection 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?
- 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?
- How did you help other coders with their projects?
 - What did you learn from other coders today?
- What's a fun algorithm you created today?
- What's something you could create next time?
- What questions do you have about coding?
 - What was challenging today?
- How did your storyboard help you plan out your project?
 - What did you end up adding that you didn't originally plan for in your storyboard?
- What is a moral in a story?
- What lessons have you learned in life that you could use as a moral for a story?
- [More sample prompts \(may need adapting for younger coders\)](#)

Sharing suggestions

Standards reinforced:

- **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development

Practices reinforced:

- Communicating about computing
- 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.