

# Character Builder

**Minimum experience:** Grades 3+, 1st year using Scratch, 2nd quarter or later

## At a Glance

### Overview and Purpose

Coders learn how to design custom costumes to create a customized character builder for a selected theme. The purpose of this project is to reinforce understandings of messages while introducing new features in Scratch.

### Objectives and Standards

#### Process objective(s):

##### Statement:

- I will learn how to create custom costumes in Scratch.

##### Question:

- How can we create custom costumes in Scratch?

#### Product objective(s):

##### Statement:

- I will create a character builder with different options for user customization.

##### Question:

- How can we create a character builder with different options for user customization?

#### Main 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 to have the character walk across the screen as they introduce themselves. ([source](#))

#### Reinforced standard(s):

**1B-AP-09** Create programs that use variables to store and modify data.

- Variables are used to store and modify data. At this level, understanding how to use variables is sufficient. For example, students may use mathematical operations to add to the score of a game or subtract from the number of lives available in a game. The use of a variable as a countdown timer is another example. ([source](#))

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

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

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

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

**1B-AP-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-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](#))
- **P5.2.** 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 1: Fostering an Inclusive Computing Culture

- "Building an inclusive and diverse computing culture requires strategies for incorporating perspectives from people of different genders, ethnicities, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products." ([p. 74](#))
- **P1.2.** Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability. ([p. 74](#))

#### 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](#))

	<ul style="list-style-type: none"> <li>● <b>P7.2.</b> Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. (<a href="#">p. 82</a>)</li> </ul>
Main concept(s):	Reinforced concept(s):
<b>Control</b> <ul style="list-style-type: none"> <li>● "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." (<a href="#">p. 91</a>)</li> <li>● <b>Grade 5</b> - "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." (<a href="#">p. 103</a>)</li> </ul>	<b>Algorithms</b> <ul style="list-style-type: none"> <li>● "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." (<a href="#">p. 91</a>)</li> <li>● <b>Grade 5</b> - "Different algorithms can achieve the same result. Some algorithms are more appropriate for a specific context than others." (<a href="#">p. 103</a>)</li> </ul> <b>Variables</b> <ul style="list-style-type: none"> <li>● "Computer programs store and manipulate data using variables. In early grades, students learn that different types of data, such as words, numbers, or pictures, can be used in different ways. As they progress, students learn about variables and ways to organize large collections of data into data structures of increasing complexity." (<a href="#">p. 91</a>)</li> <li>● <b>Grade 5</b> - "Programming languages provide variables, which are used to store and modify data. The data type determines the values and operations that can be performed on that data." (<a href="#">p. 103</a>)</li> </ul>

Scratch Blocks	
Primary blocks	<a href="#">Events</a> , <a href="#">Looks</a>
Supporting blocks	<a href="#">Control</a> , <a href="#">Variables</a> , <a href="#">Motion</a> , <a href="#">Operators</a> , <a href="#">Sound</a>

Vocabulary	
<b>Algorithm</b>	<ul style="list-style-type: none"> <li>● A step-by-step process to complete a task. (<a href="#">source</a>)</li> <li>● A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point. (<a href="#">source</a>)</li> </ul>
<b>Costume</b>	<ul style="list-style-type: none"> <li>● One out of possibly many "frames" or alternate appearances of a sprite. A sprite can change its look to any of its costumes. (<a href="#">source</a>)</li> </ul>
<b>Debugging</b>	<ul style="list-style-type: none"> <li>● The process of finding and correcting errors (bugs) in programs. (<a href="#">source</a>)</li> <li>● To find and remove errors (bugs) from a software program. Bugs occur in programs when a line of code or an instruction conflicts with other elements of the code. (<a href="#">source</a>)</li> </ul>
<b>Event (trigger)</b>	<ul style="list-style-type: none"> <li>● 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. (<a href="#">source</a>)</li> </ul>

	<ul style="list-style-type: none"> <li>The computational concept of one thing causing another thing to happen. (<a href="#">source</a>)</li> <li>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. (<a href="#">source</a>)</li> </ul>
<b>Scripts</b>	<ul style="list-style-type: none"> <li>One or more Scratch blocks connected together to form a sequence. Scripts begin with an event block that responds to input (e.g., mouse click, broadcast). When triggered, additional blocks connected to the event block are executed one at a time. (<a href="#">source</a>)</li> </ul>
<b>Variable</b>	<ul style="list-style-type: none"> <li>A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers; they can also hold text, including whole sentences (strings) or logical values (true or false). A variable has a data type and is associated with a data storage location; its value is normally changed during the course of program execution. (<a href="#">source</a>)</li> <li>Variables play an important role in computer programming because they enable programmers to write flexible programs. Rather than entering data directly into a program, a programmer can use variables to represent the data. Then, when the program is executed, the variables are replaced with real data. This makes it possible for the same program to process different sets of data. (<a href="#">source</a>)</li> </ul>

Connections	
<b>Integration</b>	<p><b>Potential subjects:</b> Art, history, language arts, media arts, music, social studies</p> <p><b>Example(s):</b> This project could include historical, contemporary, or fictional fashion and accessories, which could integrate with lessons on history, social studies, or language arts. The music player extension could also connect with music standards if coders compose or perform their own music. <a href="#">Click here</a> to see other examples and share your own ideas on our subforum dedicated to integrating projects.</p>
<b>Vocations</b>	<p>The creative thinking utilized within this project are often used by costume, fashion, or product designers. Thinking through user experience design is a common practice for software engineers (and even teachers/facilitators if we change some of the terminology). <a href="#">Click here</a> to visit a website dedicated to exploring potential careers through coding.</p>

Resources
<ul style="list-style-type: none"> <li><a href="#">Example project</a></li> <li><a href="#">Remix project</a></li> <li><a href="#">Video walkthroughs</a></li> <li><a href="#">Quick reference guides</a></li> <li><a href="#">Project files</a></li> <li><a href="#">This project was inspired by a step-by-step guide designed by Scratch</a></li> </ul>

## Project Sequence

Preparation (20+ minutes)	
Suggested preparation	Resources for learning more

**Customizing this project for your class (10+ minutes):** Remix [example project](#) or the the [remix project](#) to include your own character builder theme.

**(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.

**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 (6-15+ minutes)

### Suggested sequence

#### **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 custom costumes and code to create a character builder. If coders are not sure what character builders are, ask if anyone has played games that allow them to create their own custom character or avatar (e.g., Sims, Skyrim, and most other RPGs. Display and demonstrate the [sample project](#) (or your own remixed version).

#### **2. Discuss (3+ minutes):**

Have coders talk with each other about how they might create a project like the one demonstrated. If coders are unsure, and the discussion questions aren't helping, you can model thought processes: "I noticed the sprite moved around, so I

### Resources, suggestions, and connections

#### **Practices reinforced:**

- Communicating about computing

**Video:** [Project Preview](#) (1:33)

**Video:** [Lesson pacing](#) (1:48)

This can include a full class demonstration or guided exploration in small groups or individually. For small group and individual explorations, you can use the videos and quick reference guides embedded within this lesson, and focus on facilitating 1-on-1 throughout the process.

#### **Example review discussion questions:**

- What's something new you learned last time you coded?
  - Is there a new block or word you learned?
- What's something you want to know more about?
- What's something you could add or change to your previous project?
- What's something that was easy/difficult about your previous project?

#### **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

think they used a motion block. What motion block(s) might be in the code? What else did you notice?"

After the discussion, coders will begin working on their project as a class, in small groups, or at their own pace.

through how to create a project, are important aspects of learning to code. Not only does this process help coders think logically and creatively, but it does so without giving away the answer.

**Example discussion questions:**

- What would we need to know to make something like this in Scratch?
- What kind of blocks might we use?
- What else could you add or change in a project like this?
- What code from our previous projects might we use in a project like this?
- What kind of sprites might we see in character builder?
  - What kind of code might they have?
- How might users interact with a project like this?

**3. Remix the original project (1-10+ minutes):**

If not yet comfortable with logging in, review how to log into Scratch and remix [this project](#).

If coders continue to have difficulty with logging in, you can create cards with a coder's login information and store it in your desk. This will allow coders to access their account without displaying their login information to others.

**Alternative login suggestion:** Instead of logging in at the start of class, another approach is to wait until the end of class to log in so coders can immediately begin working on coding; however, coders may need a reminder to save before leaving or they will lose their work.

**Why the variable length of time?** It depends on comfort with login usernames/passwords and how often coders have signed into Scratch before. Although this process may take longer than desired at the beginning, coders will eventually be able to login within seconds rather than minutes.

**What if some coders log in much faster than others?** Set a timer for how long everyone has to log in to their account (e.g., 5 minutes). If anyone logs in faster than the time limit, they can open up previous projects and add to them. Your role during this time is to help out those who are having difficulty logging in. Once the timer goes off, everyone stops their process and prepares for the following chunk.

## Project Work (90+ minutes; 3+ classes)

### Suggested sequence

**4. Create custom costumes (40+ minutes)**

**2 minute discussion**

Either pick a theme for everyone, have coders select from a list of themes (see the column on the right), or allow coders to create their own theme for their projects. Discuss with a partner different ways we might create custom sprites for our theme(s).

**5+ minute demonstrations**

Drag the sprite you would like to add costumes to over Dani. Then demonstrate how to modify a costume by making a copy and then altering it with different tools in the image editor.

### Resources, suggestions, and connections

**Standards reinforced:**

- **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features

**Practices reinforced:**

- Testing and refining computational artifacts
- Creating computational artifacts

**Video:** [How to remix the Character Builder project](#) (1:27)

**Video:** [Create custom costumes](#) (2:20)

**Quick Reference Guide:** [Click here](#)



Ask for a suggestion of a new costume and create that suggestion from a blank template. Once completed, and using Dani in the stage as a guide, move the newly drawn costume around in the image editor screen (not on the stage) until it is the correct size and position on Dani on the left.

Also demonstrate how to make sprites draggable in the player view (see resources on the right).

**33+ minutes to create custom costumes and 1-on-1 facilitating**

Give coders plenty of time to create their costumes using the image editor tools. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed. If coders finish their costumes early, encourage them to begin adding code to their project.

**Video:** [Make sprites draggable](#) (1:44)

**Quick reference guide:** [Click here](#)

**Video:** [Image editor: Bitmap mode](#) (5:15)

**Video:** [Image editor: Vector mode](#) (5:00)

**Theme examples:**

- Cosplay
- Video games
- Superheros
- Anime
- Period-specific clothing (historical connection)
- Career opportunities in our community
- Future fashion design
- Funniest outfit contest

**Facilitation tip:** It might be easier to modify existing costumes than to create new costumes from scratch; use the [remix project](#) to quickly get started. Simply have coders make duplicates (click the stamp, click the costume), then modify the duplicate costume to something different.

**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.**

**(optional) Review or reverse engineering (10+ minutes each):**

If you’ve already introduced the following concepts, simply spend a few minutes asking coders how they might use the concepts in each of their projects. Otherwise, I suggest the following process for introducing how to do the following in a project.

**1+ minute intro demonstration**

Review one of the following examples:

- **Video:** [Create custom buttons](#) (2:03)
  - [Quick reference guide](#)
- **Video:** [Code your buttons](#) (2:41)
  - [Quick reference guide](#)
- **Video:** [Indicate a button is pressed](#) (2:55)
  - [Quick reference guide](#)
- **Video:** [Hold down a button](#) (2:50)
  - [Quick reference guide](#)

**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
- Testing and refining computational artifact
- Creating computational artifacts
- Fostering an inclusive computing culture

**Concepts reinforced:**

- Algorithms
- Control

- **Video:** [Reset a sprite's appearance](#) (3:41)
  - [Quick reference guide](#)

#### **4+ 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 an algorithm that makes a sprite do something similar to what was demonstrated. Facilitate by walking around and asking guiding questions.

#### **1+ minute explanation demonstration**

If coders figured out how to get their sprite 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 about how coders might change their code so it's not the same as yours.

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

**Note:** It is not recommended to show each of these ideas at once, but to show one idea, give time for application and exploration, show another idea, 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:**

- 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 events blocks in the same sprite? (*Note: this concept is Referred to as parallel computing*)
  - What are the benefits/drawbacks of using multiple events 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?

### **5. Create a character builder (40+ minutes):**

#### **40+ minute coding time and 1-on-1 facilitating**

Give coders time to add user interaction to the remix project and continue to create new costumes (as long as they also continue to add code). Encourage peer-to-peer assistance and facilitate 1-on-1 as needed.

#### **Standards reinforced:**

- **1B-AP-10** Create programs that include sequences, events, loops, and conditionals

#### **Practices reinforced:**

- Testing and refining computational artifacts
- Creating computational artifacts

#### **Concepts reinforced:**

- Algorithms
- Control

**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.



**6. Add in comments (*the amount of time depends on typing speed and amount of code*):**

**1 minute demonstration**

When the project is nearing completion, bring up some code for the project and ask coders to explain to a neighbor how the code is going to work. Review how we can use comments in our program to add in explanations for code, so others can understand how our programs work.

Quickly review how to add in comments.

**Commenting time**

Ask coders to add in comments explaining the code throughout their project. Encourage coders to write clear and concise comments, and ask for clarification or elaboration when needed.

**Standards reinforced:**

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

**Practices reinforced:**

- Communicating about computing

**Concepts reinforced:**

- Algorithms

**Video:** [Add in comments](#) (1:45)

**Quick reference guide:** [Click here](#)

**Facilitation suggestion:** One way to check for clarity of comments is to have a coder read out loud their comment and ask another coder to recreate their comment using code blocks. This may be a fun challenge for those who type fast while others are completing their comments.

## Assessment

**Standards reinforced:**

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

**Practices reinforced:**

- Communicating about computing

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

**Summative**

Assessment of Learning

The debugging exercises, commenting on code, and projects themselves can all be forms of summative assessment if a criteria is developed for each project or there are “correct” ways of solving, describing, or creating.

**For example, ask the following after a project:**

- Can coders debug the [debugging exercises](#)?
- Did coders create a project similar to the project preview?
  - **Note:** The project preview and sample projects are not representative of what all grade levels should seek to emulate. They are meant to generate ideas, but expectations should be scaled to

**Formative**

Assessment for Learning

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

**Ipsative**

Assessment as Learning

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

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 their design decisions for user interaction?
- Can coders explain how to create or modify costumes using the image editor tools?
- Did coders create a character builder with at least ## new custom costumes and at least ## different ways for users to interact with their project?
  - Choose a number appropriate for the coders you work with and the amount of time available.

[assessment examples](#) for more questions.

the end for more suggestions.

## Extended Learning

### Project Extensions

#### Suggested extensions

##### **Use the example project as a guide (as needed)**

At some point, coders might get stuck or run out of ideas. Rather than explaining to them how to do something, ask them to open the [example project](#), read the comments inside the various sprites and background, and then look at the code to see if they can figure out how to solve their problem. Although this is a very open-ended approach, this models a common coding practice that helps coders become independent learners.

#### Resources, suggestions, and connections

##### **Standards reinforced:**

- **1B-AP-10** Create programs that include sequences, events, loops, and conditionals
- **1B-AP-12** Modify, remix, or incorporate portions of an existing program into one's own work, to develop something new or add more advanced features

##### **Practices reinforced:**

- Testing and refining computational artifacts
- Creating computational artifacts

##### **Concepts reinforced:**

- Algorithms
- Control

**Resource:** [Example project](#)

**Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the [Tutorials](#) to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions.

	<p><b>Note:</b> The example project has a lot of messages that all do different things. It makes it look like the project is complicated; however, each message/function is relatively easy to understand and has comments that explains what it does.</p>
<p><b><u>Match sounds to a costume (15+ minutes)</u></b>  <b>5 demonstration</b>  Sometimes it makes sense to have many different kinds of costumes in one sprite (e.g., the Props sprite); however, what happens if we want that sprite to play a different sound depending on the costume it's on? Demonstrate how the Props sprite will play different sounds that match the different costumes, then ask the class how we might do this. This is pretty tricky as it introduces a block we haven't used yet, so I wouldn't expect them to know the answer.</p> <p>Demonstrate how to match sounds to a costume number using the <a href="#">looks costume # block</a> inside of a <a href="#">play/start sound block</a>.</p> <p><b>10+ minutes to code their projects and 1-on-1 facilitating</b>  Give coders time to match sounds to costumes and trigger them with code. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed.</p>	<p><b>Standards reinforced:</b></p> <ul style="list-style-type: none"> <li>● <b>1B-AP-10</b> Create programs that include sequences, events, loops, and conditionals</li> </ul> <p><b>Practices reinforced:</b></p> <ul style="list-style-type: none"> <li>● Testing and refining computational artifacts</li> <li>● Creating computational artifacts</li> </ul> <p><b>Concepts reinforced:</b></p> <ul style="list-style-type: none"> <li>● Algorithms</li> <li>● Control</li> </ul> <p><b>Video:</b> <a href="#">Match sounds to a costume</a> (3:17)  <b>Quick reference guide:</b> <a href="#">Click here</a></p>
<p><b><u>Create music player controls (Advanced) (5+ minutes per function)</u></b>  <b>2 minute demonstration for each function</b>  To create a music player, we need to create a variable to keep track of what song we are on. Demonstrate how to create a variable in the backdrops section, and then how to use the message blocks to create functions for play, next song, previous song, and stop (demonstrate one at a time).</p> <p><b>3+ minutes coding time for each function, then time at the end for tinkering and 1-on-1 facilitating</b>  Give coders time to code each function of their music players. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed.</p>	<p><b>Standards reinforced:</b></p> <ul style="list-style-type: none"> <li>● <b>1B-AP-09</b> Create programs that use variables to store and modify data</li> <li>● <b>1B-AP-10</b> Create programs that include sequences, events, loops, and conditionals</li> </ul> <p><b>Practices reinforced:</b></p> <ul style="list-style-type: none"> <li>● Testing and refining computational artifacts</li> <li>● Creating computational artifacts</li> </ul> <p><b>Concepts reinforced:</b> <i>Algorithms, Control, and Variables</i></p> <p><b>Video:</b> <a href="#">Create music player controls (Advanced)</a> (6:01)  <b>Quick reference guide:</b> <a href="#">Click here</a></p> <p><b>Note:</b> Keeping music players in the backdrops area makes the music files and code easier to find than having to search for the code in every single sprite. A quick way to remember where to store it is to put the background music's code in the background (backdrop).</p> <p><b>Why is this labeled advanced if the code is so simple?</b> The code itself is very simple to create; however, we haven't talked about how to use variables yet. We will introduce variables in later projects.</p>
<p><b><u>Add even more (30+ minutes, or at least one class):</u></b>  If time permits and coders are interested in this project, encourage coders to explore what else</p>	<p><b>Standards reinforced:</b></p> <ul style="list-style-type: none"> <li>● <b>1B-AP-10</b> Create programs that include sequences, events, loops, and conditionals</li> </ul> <p><b>Practices reinforced:</b></p>

they can create in Scratch by trying out new blocks and reviewing previous projects to get ideas for this project. When changes are made, encourage them to alter their comments to reflect the changes (either in the moment or at the end of class).

While facilitating this process, monitor to make sure coders don't stick with one feature for too long. In particular, coders like to edit their sprites/backgrounds by painting on them or taking photos, or listen to the built-in sounds in Scratch. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.

#### **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.

- Testing and refining computational artifacts
- Creating computational artifacts

#### **Concepts reinforced:**

- Algorithms
- Control

**Facilitation Suggestion:** Some coders may not thrive in inquiry based approaches to learning, so we can encourage them to use the [Tutorials](#) to get more ideas for their projects; however, we may need to remind coders the suggestions provided by Scratch are not specific to our projects, so it may create some unwanted results unless the code is modified to match our own intentions.

#### **Suggested questions:**

- What else can you do with Scratch?
- What do you think the other blocks do?
  - a. Can you make your project do \_\_\_\_?
- What other sprites can you add to your project?
- What have you learned in other projects that you could use in this project?
- Can you add more user control than demonstrated?
- How else might you use messages in your 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

#### **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?
- What kind of buttons could we add to this project?
- How could we use messages in this project?
- How might a user interact with this project in ways that we currently cannot?

<p><b>micro:bit extensions:</b></p> <p><b>Note:</b> the micro:bit requires installation of Scratch Link and a HEX file before it will work with a computer. Watch <a href="#">this video</a> (2:22) and <a href="#">use this guide</a> to learn how to get started with a micro:bit before encouraging coders to use the <a href="#">micro:bit blocks</a>.</p> <p>Much like the generic <a href="#">Scratch Tips folder</a> linked in each Coder Resources document, the <a href="#">micro:bit Tips folder</a> contains video and visual walkthroughs for project extensions applicable to a wide range of projects. Although not required, the <a href="#">micro:bit Tips folder</a> 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.</p>	<p><b>Standards reinforced:</b></p> <ul style="list-style-type: none"> <li>● <b>1B-AP-09</b> Create programs that use variables to store and modify data</li> <li>● <b>1B-AP-10</b> Create programs that include sequences, events, loops, and conditionals</li> <li>● <b>1B-AP-11</b> Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process</li> <li>● <b>1B-AP-15</b> Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended</li> </ul> <p><b>Practices reinforced:</b></p> <ul style="list-style-type: none"> <li>● Recognizing and defining computational problems</li> <li>● Creating computational artifacts</li> <li>● Developing and using abstractions</li> <li>● Fostering an inclusive computing culture</li> <li>● Testing and refining computational artifacts</li> </ul> <p><b>Concepts reinforced:</b></p> <ul style="list-style-type: none"> <li>● Algorithms</li> <li>● Control</li> <li>● Modularity</li> <li>● Program Development</li> <li>● Variables</li> </ul> <p><b>Folder with all micro:bit quick reference guides:</b> <a href="#">Click here</a></p> <p><b>Additional Resources:</b></p> <ul style="list-style-type: none"> <li>● Printable micro:bit cards <ul style="list-style-type: none"> <li>○ <a href="#">Cards made by micro:bit</a></li> <li>○ <a href="#">Cards made by Scratch</a></li> </ul> </li> <li>● <a href="#">Micro:bit's Scratch account with example projects</a></li> </ul> <p><b>Generic questions:</b></p> <ul style="list-style-type: none"> <li>● How can you use a micro:bit to add news forms of user interaction?</li> <li>● What do the different <a href="#">micro:bit event blocks</a> do and how could you use them in a project?</li> <li>● How could you use the LED display for your project?</li> <li>● What do the <a href="#">tilt blocks</a> do and how could you use them in your project?</li> <li>● How could you use the buttons to add user/player controls?</li> <li>● How might you use a micro:bit to make your project more accessible?</li> </ul>
---	---

Differentiation	
Less experienced coders	More experienced coders
<p>Demonstrate the example <a href="#">remix project</a> or your own version, and walk through how to experiment changing various parameters or blocks to see what they do. Give some time for them to change the blocks around. When it appears a coder might need some guidance or has completed an idea, encourage them to add more to the project or begin following the steps for creating the project on their own (or with BootUp resources). Continue to facilitate one-on-one using</p>	<p>Demonstrate the project without showing the code used to create the project. Challenge coders to figure out how to recreate a similar project without looking at the code of the original project. If coders get stuck reverse engineering, use guiding questions to encourage them to uncover various pieces of the project. Alternatively, if you are unable to work with someone one-on-one at a time of need, they can access</p>

questioning techniques to encourage tinkering and trying new combinations of code.

If you are working with other coders and want to get less experienced coders started with remixing, have those who are interested in remixing a project [watch this video](#) (2:42) to learn how to remix a project.

the quick reference guides and video walkthroughs above to learn how each part of this project works.

If you are working with other coders and want to get more experienced coders started with reverse engineering, have those who are interested [watch this video](#) (2:30) to learn how to reverse engineer a project.

## Debugging Exercises (1-5+ minutes each)

### Debugging exercises

[Why doesn't the bowtie spin when clicked?](#)

- [The second "turn" block makes it return back to where it started, and when the algorithm runs very fast, it makes it appear like it's not moving. Change it so there's only one "turn" block.](#)

[Why does the "next song" button work, but the "previous song" button keeps repeating the same song?](#)

- [Our "Song number" variable needs to change by -1, not set to -1](#)

[Why doesn't the shirt move to Dani when the random button is clicked?](#)

- [We need to change the parameters of the "glide" block to the numbers that correspond with Dani's location.](#)

**\*micro:bit required\*** [Why can I reset and randomize the costumes with the micro:bit buttons, but only Dani is randomized when I shake the micro:bit?](#)

- [We need to add the "broadcast random" message block in order to trigger the "when I receive random" in each of the costume sprites](#)

[Even more debugging exercises](#)

### Resources and suggestions

**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?

## 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

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?
- 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?
- When should you use buttons and when should you use keyboard commands?
  - What about using both?
  - How might either approach make it easier or harder for people to use your project?
- How might you customize costumes and backdrops in previous or future projects?
- How are the image editor tools similar or different from other programs you've used?
- [More sample prompts](#)

### Sharing suggestions

#### 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

#### 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](#)