Unit 6 - Physical Computing

In Unit 6, students further develop their programming skills, while exploring more deeply the role of hardware platforms in computing. Harkening back to the Input/Storage/Processing/Output model for a computer, students look towards modern “smart” devices to understand the ways in which non-traditional computing platforms take input and provide output in ways that couldn't be done with the traditional keyboard, mouse, and monitor.

Using App Lab and Adafruit’s Circuit Playground, students develop programs that utilize the same hardware inputs and outputs that we see in many modern smart devices, and they get to see how a simple rough prototype can lead to a finished product. The unit concludes with a design challenge that asks students to use the Circuit Playground as the basis for an innovation of their own design.

Chapter 1: Programming with Hardware

Big Questions

- How does software interact with hardware?
- How can computers sense and respond to their environment?
- What kind of information can be communicated with simple hardware outputs?

Week 1

Lesson 1: Innovations in Computing

Research | Unplugged

Explore a wide variety of new and innovative computing platforms while expanding your understanding of what a computer can be.

Lesson 2: Designing Screens with Code

App Lab

By reading and changing the content on the screen of an app, the class starts to build apps that only need a single screen. Even with just one screen, these techniques allow for lots of user interaction and functionality.

Lesson 3: The Circuit Playground

App Lab | Maker Toolkit

Get to know the Circuit Playground, the circuit board that will be used throughout the rest of this unit. Using App Lab, develop programs that use the Circuit Playground for output.

Week 2

Lesson 4: Input Unplugged

Unplugged

Experience two different ways that an app can take input from a user, while learning more about the event-driven programming model used in App Lab.

Lesson 5: Board Events

App Lab | Maker Toolkit

Using the hardware buttons and switch, the class develops programs that use the Circuit Playground as an input.

Lesson 6: Getting Properties

App Lab | Maker Toolkit

This lesson introduces students to the getProperty block, which allows them to access the properties of different elements with code. Students first practice using the block to determine what the user has input in various user interface elements. Students later use getProperty and setProperty together with the counter pattern to make elements move across the screen. A new screen element, the slider, and a new event trigger, onChange, are also introduced.

Lesson 7: Analog Input

App Lab | Maker Toolkit

Explore the analog inputs on the Circuit Playground, writing programs that respond to the environment through sensors.

Week 3

Lesson 8: The Program Design Process

App Lab | Maker Toolkit

This lesson introduces students to the process they will use to design programs of their own throughout this unit. This process is centered around a project guide which asks students to sketch out their screens, identify elements of the Circuit Playground to be used, define variables, and describe events before they begin programming. This process is similar to the Game Design Process that we used in Unit 3. In this lesson students begin by playing a tug o' war style game where the code is hidden. They discuss what they think the board components, events, and variables would need to be to make the program. They are then given a completed project guide which shows one way to implement the project. Students are then walked through this process through a series of levels. At the end of the lesson students have an opportunity to make improvements to the program to make it their own.

Lesson 9: Project: Make a Game

App Lab | Maker Toolkit | Project

Students take what they've learned through chapter one, and develop an app of their own design that uses the board to output information.

Chapter Commentary

This unit begins with an activity that encourages students to explore a wide variety of non-traditional computing platforms, before kicking off a review of programming in App Lab, with a particular focus on better understanding the event-driven programming model that was first introduced in Unit 4. Students learn techniques to make the apps they write more flexible by modifying design elements through code instead of always relying on design

mode. Using the Circuit Playground, they then explore different approaches to taking input and producing output using hardware. By the end of this chapter, students will design a develop a game that uses physical hardware for input and output.

Chapter 2: Building Physical Prototypes

Big Questions

- How do programmers work with larger amounts of similar values?
- How can complex real-world information be represented in code?
- How can simple hardware be used to develop innovative new products?

Week 4

Lesson 10: Arrays and Color LEDs

App Lab | Maker Toolkit

Learn how lists can be used to store multiple values in a single variable name.

Lesson 11: Making Music

App Lab | Maker Toolkit

In this lesson students will use the buzzer to its full extent by producing sounds, notes, and songs with the buzzer. Students start with a short review of the buzzer's frequency and duration parameters, then move on to the concept of notes. Notes allow students to constrain themselves to frequencies that are used in Western music and provide a layer of abstraction that helps them to understand which frequencies might sound good together. Onces students are able to play notes on the buzzer, they use arrays to hold and play sequences of notes, forming simple songs.

Lesson 12: Arrays and For Loops

App Lab | Maker Toolkit

Combining _lists_ and _for loops_ allows you to write code that impacts every element of a list, regardless of how long it is. The class uses this structure to write programs that process all of the elements in lists, include the list of color LEDs.

Lesson 13: Accelerometer

App Lab | Maker Toolkit

In this lesson, students will explore the accelerometer and its capabilities. They'll become familiar with its events and properties, as well as create multiple programs utilizing the accelerometer similar to those they've likely come across in real world applications.

Week 5

Lesson 14: Functions with Parameters

App Lab | Maker Toolkit

The lesson starts with a quick review of parameters, in the context of App Lab blocks that they students have seen recently. Students then look at examples of parameters within user-created functions in App Lab and create and call functions with parameters for themselves, using them to control multiple elements on a screen. Afterwards, students use for loops to iterate over an array, passing each element into a function. Last, students use what they have learned to create a star catching game.

Lesson 15: Circuits and Physical Prototypes

App Lab | Maker Toolkit

Wire simple circuits to create a physical prototype using cheap and easily found materials.

Week 6

Lesson 16: Project: Prototype an Innovation

App Lab | Maker Toolkit | Project

Develop innovative computing devices of your own design, using everything you've learned throughout this course.

Chapter Commentary

With an understanding of how to use hardware to take input and produce output, students move to thinking about more complex programs that integrate hardware and software. Using the color LEDs as an example of a a group of like objects, students learn how to use arrays to keep track of lists of values. From there we introduce the *for loop*, first simply as a way to repeat a block of code, and then as a way to run code on each element of an array. By the end of this chapter students will have explored all components of their boards while learning to structure their code using arrays, loops, and parametric functions. In the final two lessons students have an opportunity to dig into building physical prototypes using their boards.

Lesson 1: Innovations in Computing

Overview

To kick off the final unit of this course, students will do some research into interesting innovations in computing. This lesson will expose students to wider variety of computing form factors (what a computer looks like) and fields that are impacted by computing. Later in this unit students will look back on the devices they encountered in this lesson as they develop their own physical computing devices.

Purpose

This lesson will lay the groundwork for students' understanding of how their Circuit Board could be used to model an innovative computing device. The goal is to get them thinking about how computers can be embedded into just about anything, and to start considering the potential impacts of such applications.

Agenda

Warm Up (10 min)
- **Get Inspired**

Activity (45 min)
- **Innovation Research**

Wrap Up (2 min)
- **Journaling**

Extensions
- **Innovation Posters**
- **Innovation Websites**

View on Code Studio

Objectives

Students will be able to:

- Identify computing innovations within a given field
- For a given device, articulate the likely inputs and outputs
- Suggest improvements to help a device better solve a specific problem

Preparation

- [ ] Review the resource pages linked in Code Studio
- [ ] Cue up **The Internet of Things - Video** or **Computer Science is Changing Everything - Video**
- [ ] Print out a copy of **Computing Innovations - Activity Guide** for each student

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Computing Innovations** - Activity Guide Make a Copy
- **The Internet of Things** - Video
- **Computer Science is Changing Everything** - Video (**download**)

Vocabulary

- **Innovation** - A new or improved idea, device, product, etc, or the development thereof

Teaching Guide

Warm Up (10 min)

Get Inspired

Video: Watch either **The Internet of Things - Video** or **Computer Science is Changing Everything - Video** as a class.

Teaching Tip

This unit requires the Adafruit Circuit Playground. You can read more about this microcontroller board at **https://code.org/circuitplayground**

Activity (45 min)

Innovation Research

Group: Place students in groups of 3-4

Distribute: Hand out copies of **Computing Innovations - Activity Guide**

Discuss: What is an innovation? What does it mean for something to be innovative?

Discussion Goal

This should be a quick discussion. The primary goal is for the class to come to a common understanding of what it means for something to be innovative. That it's not enough for a product to look sleeker than the last version, but that innovation means to really do something better than it's been done before, or to do things that have *never* been done before.

Computing Innovations

During this activity student groups will research the recent technological innovations related to a chosen topic. Once they have identified a few interesting innovations, they will choose one to analyze in greater depths and report back to the class about.

Innovation Research

- **Introduce the topics**: Make sure that students understand the scope of each of the potential topics.
  - Wearable Technology (eg. clothing, jewelry, or accessories with built-in computers)
  - Health and Safety (eg. devices that treat disease, track your health, or protect users from danger)
  - Agriculture (eg. technology to improve the effectiveness, sustainability, or efficiency of farming)
  - Manufacturing (eg. advancements in rapid prototyping, industrial robotics, and the production of goods)
  - Art and Design (eg. interactive art or public installations)
  - Smart Home (eg. devices that allow you to interact with your thermostat, locks, or lights using computers)
- **Explain the research task**: The goal of this research is twofold:
  - First, develop a deeper understanding of your chosen topic. How is computer technology changing this field, what are some of the problems that people are trying to solve with technology?
  - Second, identify a handful of innovative devices within this topic. Students should focus on finding *hardware* devices that demonstrate unique or novel *form factors*. That is to say, computers that don't *look* like computers.
- **Send to Code Studio for resource links**: On Code Studio we have compiled more detailed descriptions of the topics as well as couple of recommended sites to learn more about each topic. Use this as a jumping off point for student research.

Code Studio levels

Lesson Overview | Student Overview

Innovation Research | Teacher Overview | Student Overview

Here are some recommended sites for students to start their research. Although these sites are generally appropriate for school, the content within them changes frequently, so **we strongly suggest you check each site for inappropriate content before sharing it with students**.

View on Code Studio

Wearable Technology

- **Warable.com**

Health and Safety

- **Modern Healthcare**
- **Medstartr**

Agriculture

- **National Institute of Food and Agriculture**
- **Farm Industry News**

Manufacturing

- **Industry Week**
- **3D Printing**

Art and Design

- **ArtFab**
- **Instructables**

Smart Home

- **CNet**
- **IoT Evolution**

An Innovative Solution

Once groups have selected a specific innovative device, they can complete the second page of this activity guide, which asks them to do the following:

- **What Problem Does It Solve?:** For some topics this may be more clear (for example healthcare devices typically have a very specific goal), but for others students may need to think more broadly about what they consider problems. More frivolous wearables or digital art installations may be more concerned with personal expression than solving a specific problem.
- **What Is Innovative About It?:** Encourage students to refer back to the earlier discussion about innovation for this. It can also be useful to compare against other devices found during the research phase.
- **How Do You Interact With It?:** Ask students to think back to the *Input, Output, Storage, Processing* model that was introduced way back in Unit 1. The goal here is to identify specifically the Inputs and Outputs provided by a given device - the more specific these are, the more easily students will be able to connect elements of the Circuit Playground to these devices.
- **How Could You Improve It?:** Feel free to make this as realistic or aspirational as you like. The goal here is just to get students thinking about how they might develop an innovation of their own by using an existing product as a jumping off point.

Share: Give groups a minute each to share their findings. See the extension activities for alternate ways to share.

Wrap Up (2 min)

Journaling

Journal: What was the most surprising, cool, or impressive thing that you found in your research? If you could develop an innovation of your own, what would it be?

Extensions

Innovation Posters

Ask groups to create a poster that shares the innovate device that they researched. Posters should include:

- Who invented the device
- What problem they were trying to solve
- Why it is unique or innovative
- How users interact with it (specifically, what Inputs does it take and how does it provide Output)

Keep these posters up throughout the unit for students to refer back to as they start to develop physical computing devices of their own.

Innovation Websites

Instead of (or in addition to) you posters, have students develop websites in Web Lab that include the same required content. Make sure that students include links to their source websites.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **IC** - Impacts of Computing

Lesson 2: Designing Screens with Code

Overview

In Unit 4 students learned a very simple approach to app development in App Lab that required a separate screen for most interactions. To expand the kinds of apps that students can make, and to encourage them to think in new ways about how users interact with apps, we introduce the `setProperty()` block. This command can be used to set the content and properties of various UI elements, allowing students to write programs that update information on a single screen, instead of manually creating duplicate screens. In this lesson students build up simple apps that only require a single screen, the content of which is changed using `setProperty()`.

Purpose

This lesson allows students some time to get back into programming with App Lab before introducing the Circuit Playground, but also introduces the useful concept of a *setter*. In Unit 4, students primarily used `setScreen()` to make their apps respond to user interaction. While this is a simple and useful technique, it can lead to a lot of duplication of content across multiple screens. By using *setters*, and later *getters*, students can write apps that actually change the content on a single screen, by showing and hiding or changing the content or look of various elements.

Once students have learned about using getters and setters with UI elements, encourage them to think critically about when to use a separate screen in the design phase of an app. If screens are more alike than different, it might be more effective to just change the elements on screen in reaction to input instead of duplicating content across multiple screens. While students are only introduced to `setProperty()` right now, they will later learn the partner command `getProperty()`.

Agenda

Warm Up (5 min)

- **UI Element Properties Refresher**

Activity (45 to 60 minutes)

- **Designing with setProperty()**

Wrap Up (5 min)

- **Reflecting on Unit 4**

View on Code Studio

Objectives

Students will be able to:

- Set the properties of UI elements using code
- Respond to user input using an event handler
- Write programs that change multiple elements on a single screen instead of changing screens

Introduced Code

- `setProperty(id, property, value)`

Teaching Guide

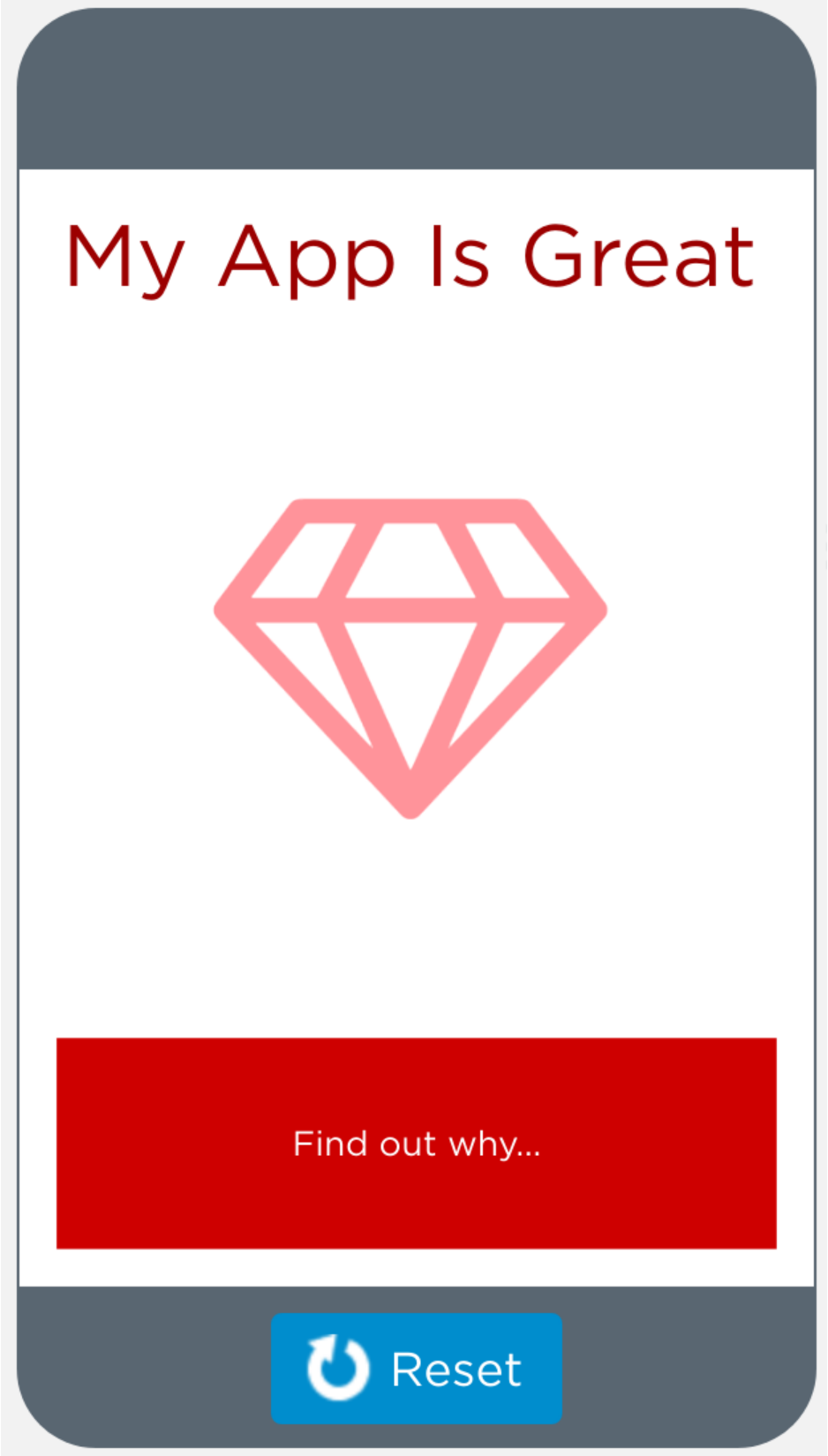
Warm Up (5 min)

UI Element Properties Refresher

Display: Show students this example image

Discussion Goal

The goal of this disucssion is to prime students to think about the *properties* of various design elements, which they will learn to change with code later in this lesson. You might ask students to think back to a similar discussion we had in Unit 2 when trying to describe what a web page looks like, or to the properties of a sprite from Unit 3



Discuss: How would you describe to somebody else how to recreate this screen? What specific details would they need to know about each design element?

Activity (45 to 60 minutes)

Designing with setProperty()

Transition: Head to Code Studio

Code Studio levels

Lesson Overview | Student Overview

Designing Screens with Code | Student Overview

Setting Properties | 3 | 4 | 5 *(click tabs to see student view)*

Events and Properties | 6 | 7 | 8 | 9 *(click tabs to see student view)*

Emotion Machine Example | Student Overview

Building an App | 11 | 12 | 13 | 14 *(click tabs to see student view)*

Share: The final level in this lesson allows students to customize and submit an "Emotion Machine" app. If time, allow students to share their programs.

Wrap Up (5 min)

Reflecting on Unit 4

Prompt: Think back to the app you prototyped in Unit 4. Knowing what you know about using `setProperty()` to change UI elements, how might you change your app prototype?

Discuss: Have the class share the kinds of things they came up with. See if the class can come up with some broad types of features that weren't possible, or were cumbersome, to do with screens alone.

Discussion Goal

Goal: This discussion is intended to clarify for students *why* we are changing UI elements with code, and how it might actually allow them to solve problems that came up in Unit 4. From this point on students will need to make reasoned choices about when to use separate screens in a program and when to update elements with code.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

Lesson 3: The Circuit Playground

Overview

In this lesson students get their first opportunity to write programs that use the Circuit Playground. After first inspecting the board visually and hypothesizing possibly functionalities, students move online where they will learn to write applications that control an LED. By combining App Lab screens with the Circuit Playgrounds, students can gradually start to integrate elements of the board as an ouput device while relying on App Lab for user input.

Purpose

As the first introduction to using the Circuit Playground, this lesson leaves time for students to get comfortable with getting the hardware plugged in. By leveraging students' existing knowledge of event handling in App Lab, we can quickly get an app up and running that shows the potential of physical computing with little more than a single red LED.

Agenda

Warm Up (5 min)

- **Board Inspection**

Activity (45)

- **Connecting the Board**
- **Programming on Hardware**

Wrap Up (5 min)

- **What's in a Board**

View on Code Studio

Objectives

Students will be able to:

- Connect and troubleshoot external devices
- Turn on and off an LED with code
- Use code to control a physical device

Preparation

- [ ] Make sure that student computers have the drivers and software necessary to connect to the Circuit Playground (**details here**)
- [ ] Prepare a board and USB cable for each pair of students

Introduced Code

- `led.on()`
- `led.off()`
- `led.blink(interval);`
- `led.pulse(interval);`
- `led.toggle()`

Teaching Guide

Warm Up (5 min)

Board Inspection

Distribute: Pass out a board and USB cable to each pair of students. Let students know that they should not yet plug the boards in.

Prompt: Ask pairs to spend one minute looking over the board, focusing on the details. What do you think this board does (or could do) and why?

Share: Have groups share back their thoughts to the whole group, keeping track of ideas on the board. Push students to support their ideas with evidence from reviewing the board, but don't worry about ensuring correctness at this point. As students go through this unit, they can refer back and refine this list.

Activity (45)

Connecting the Board

Transition: Ask students to plug their boards in and head to **the Maker Toolkit setup page** to confirm that the software has been correctly configured.

Programming on Hardware

Transition: Send students to Code Studio

Code Studio levels

Lesson Overview | Student Overview

Hardware and Software | Student Overview

Circuit Playground: LEDs | Student Overview

Using the LED | 4 | 5 | 6 | 7 | 8 | 9

(click tabs to see student view)

LED Apps | 10 | 11 | 12 | 13 | 14 ***(click tabs to see student view)***

Wrap Up (5 min)

What's in a Board

Journal: Ask students to reflect on their introduction to the Circuit Playground. What did they think it was at first inspection? How did those expectations change after having programmed on the board?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems

Lesson 4: Input Unplugged

Overview

In preparation for delving deeper into programming with App Lab, students will explore how a handful of different programs written in both Game Lab and App Lab handle taking input from the user. After comparing and contrasting the approaches they saw in the example apps, students group up to act out the two different models for input (conditionals in an infinite loop and asynchronous events) to gain a better understanding of how they work.

Purpose

This lesson is intended to help students transition from the *draw loop* and conditional model of input used in Game Lab to the event-driven model used in App Lab. While students have experienced and learned a bit about event-driven programming in Unit 4, a deeper understanding of how the events work will help when it comes to responding to events on the Circuit Playground later in this unit.

Agenda

Warm Up (10 min)

Comparing Input Methods

Activity (40 min)

Input Unplugged
Activity Debrief

Wrap Up (2 min)

Reflection

View on Code Studio

Objectives

Students will be able to:

- Compare and contrast multiple ways to take input
- Describe the elements of an event handler
- Model different methods of taking user input

Preparation

- [ ] Prepare to display example programs for the whole class.
- [ ] A half deck of cards for each group of three students or **deck-of-cards.js.org**.
- [ ] Print one copy of **Input and Events - Activity Guide** for each group of three students.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Input and Events** - Activity Guide
  Make a Copy

Teaching Guide

Warm Up (10 min)

Comparing Input Methods

Discuss: Yesterday we wrote some apps that took input from a user, but how did the program know *when* to take input?

Display: Open up the Code Studio stage for this level. For each of the example apps, ask the class to identify:

1. Where is the input coming from? (e.g. keyboard, mouse, etc)
2. What input value is the program looking for?
3. How will the program respond to input?

Discussion Goal

Goal: This is intended to be a very broad question, and as such could go in a number of different directions. If your students are struggling, encourage them to think about specific programs that they wrote in units 3 and 4.

Code Studio levels

Lesson Overview | **Student Overview**

Input Examples | 2 | 3 | ***(click tabs to see student view)***

Prompt: Ask the class the following questions, keeping track of answers on the board. The goal at this point isn't to answer questions that come up, but to record them so we can see how the class feels again after the lesson.

- How did the different apps we looked at approach taking input similarly or differently?
- Which way made the most sense to you?
- Which way made the least sense?
- What questions to you have about how these programs take input?

Activity (40 min)

Input Unplugged

Remarks

> In the previous activity we explored several different kinds of input, but how do our programs actually process that input? As a class, and in small groups, we're going to model how input is taken both in Game Lab and App Lab.

Group: Place students in groups of 4

Distribute: Hand out a copy of **Input and Events - Activity Guide** and half of a deck of cards to each group (the deck doesn't need to be perfectly split, but should include both red and black cards in each half). The activity guide includes four different role pages, printed front and back.

Model: Choose one group to model the activity for the class.

Input and Events

> The goal of this activity is to help students better understand how events work, and how using events differs from the input model that they used in Game Lab. In groups of four, students will model the two different approaches to taking input. By the end of the activity students should understand that the `onEvent` block sets up a input to

watch for events in the background, allowing input handling without explicitly asking each input for a value constantly. The roles for this activity are:

Program: The Program reads the code aloud and performs any actions dictated by the inputs. **Inputs 1-3:** The Inputs draw cards from the deck, which represent their changing input values.

Version A: Asking for Input

This first version models the Game Lab approach to taking input. In this scenario the **Program** explicitly asks each input for a value every time the draw loop is run.

Rules for Version A:

- The **Program** reads each line of code aloud.
- At the beginning of each draw loop, each **Input** draws a new card from the deck. This represents the current state of that input.
- Whenever the **Program** reaches an input checking command (`inputOne()` , `inputTwo()` , or `inputThree()`), the **Program** asks to see that inputs card.
- If the value of the card shown matches the associated conditional, the **Program** performs the action(s) inside the conditional.

Teaching Tip

The programs for both versions of this activity are purposely very simple to allow students to focus on the processes involved in each input method. If your students pick up on the ideas quickly, consider providing some more detailed example programs to push on their understanding. In particular, placing the inputs or events in a different order and nesting conditionals can be useful in revealing misconceptions.

After groups have run the program a few times and seem to understand how it works, have them switch up roles so that each student gets a chance to be the **Program**.

Version B: Input Events

This time around, instead of constantly asking for input in a loop, the **Program** will assign each **Input** an Event to watch for. In the context of this activity, an Event is either a red or black card, but in a program that Event could be a click on a button, movement of the mouse, press of a key, or more. This is a simplified model for the way input is handled in App Lab.

Rules for Version B:

- Each **Input** draws a card from the deck at a rate of roughly one per second.
- The **Program** reads each line of code aloud.
- When the **Program** reaches an `onEvent` command, they tell the specified **Input** which Event to watch for, and what the Response should be.
- When assigned an Event, the **Input** writes down the details in their *Events to Watch* table.
- Every time an **Input** draws a card, they check to see if it matches one of the Events in their table. If it does, they tell the **Program** to perform the Response.

There are a few important simplifications that are made in this model to minimize the number of roles and rules:

- This model implies that Inputs are actually watching for Events on their own. In reality, an Event Handler running in the background watching for the specified event.
- In this model the **Input** tells the **Program** what actions to take. In reality, when an Event Handler is triggered, the Program goes to that portion of their code and runs everything in the function.

Activity Debrief

Discuss: Looking back at the comments gathered from the warm up activity, what things are more clear after having run the activity? Do you have any new questions?

Wrap Up (2 min)

Reflection

Journal: What's the difference between the way that Game Lab and App Lab handle inputs?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

Lesson 5: Board Events

Overview

This lesson transitions students from consider the Circuit Playground as strictly an output device towards using it as a tool for both input and output. Starting with the hardware buttons and switch,sing the hardware buttons and switch, students learn to use `onBoardEvent()`, analogously to `onEvent()`, in order to take input from their Circuit Playgrounds.

Purpose

This lesson marks the transition from using the board solely as an output device to using it for both input and output. The `onBoardEvent()` block works much like `onEvent()`, with the most significant different being that the first parameter is a board object (a variable) while `onEvent()` takes a UI element ID (a string).

Agenda

Warm Up (5 min)

Board Inspection: Inputs

Activity (45 min)

Taking Input from the Board

Wrap Up (2 min)

View on Code Studio

Objectives

Students will be able to:

- Attach an event handler to a hardware input
- Choose the appropriate event for a given scenario

Introduced Code

- `onBoardEvent(component, event, function(event) {...});`
- `toggleSwitch`
- `toggleSwitch.isOpen`
- `button(L/R)`
- `buttonL.isPressed`

Teaching Guide

Warm Up (5 min)

Board Inspection: Inputs

Distribute: Pass out a board and USB cable to each pair of students. Let students know that they should not yet plug the boards in.

Prompt: Ask pairs to spend one minute looking over the board, focusing on potential input devices. Based on what you already know about this board, how do you think you might use it to get input?

Share: Have groups share back their thoughts to the whole group, keeping track of ideas on the board. Push students to support their ideas with evidence from reviewing the board, but don't worry about ensuring correctness at this point.

Activity (45 min)

Taking Input from the Board

Transition: Send students to Code Studio.

Code Studio levels

Lesson Overview — Student Overview

Responding to Board Events — Student Overview

Board Events — 3 | 4 | 5 *(click tabs to see student view)*

Buttons and Switch — Student Overview

Using the Switch — 7 | 8 | 9 *(click tabs to see student view)*

Circuit Playground: Buzzer — Student Overview

The Buzzer — 11 | 12 | 13 *(click tabs to see student view)*

Wrap Up (2 min)

Journal: Given the types of board events that you saw today, how might you take input from other elements on the board? What other events do you think might exist?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming
- ▶ **CS** - Computing Systems

Lesson 6: Getting Properties

Overview

This lesson introduces students to the `getProperty` block, which allows them to access the properties of different elements with code. Students first practice using the block to determine what the user has input in various user interface elements. Students later use `getProperty` and `setProperty` together with the counter pattern to make elements move across the screen. A new screen element, the slider, and a new event trigger, `onChange`, are also introduced.

Purpose

So far, students have only used the button press event to gather information from the App Lab screen. Using `getProperty` opens the students to gathering a wide range of information from the user. Getting properties also allows students to create programs that don't rely on knowing an element's properties when the code is written, so their programs can be more dynamic and interactive. These features will be important as students move toward making an interactive game in the next few lessons.

Agenda

Warm Up (5 min)
Activity (40 min)
Wrap Up (10 min)

View on Code Studio

Introduced Code

- `getProperty(id, property)`

Teaching Guide

Warm Up (5 min)

Review: Ask students to think of the different types of information that they were able to get from the board in the last lesson (e.g. button presses, toggle state, etc.)

Remarks

> So far we've seen lots of ways that we can get information from our Circuit Playground. Today we're going to look at more ways that we can get information from the screen of our app.

Activity (40 min)

Send students to Code Studio.

Code Studio levels

Lesson Overview Student Overview

getProperty Student Overview

Getting Properties 3 4 5 6 7 *(click tabs to see student view)*

Sliders 8 9 10 *(click tabs to see student view)*

Event Types Student Overview

Change 12 13 14 *(click tabs to see student view)*

Motorcycle 15 16 *(click tabs to see student view)*

Wrap Up (10 min)

Journal Prompt: So far, you've seen several different types of input, some from the screen, and some from the circuit playground. Choose one type of input and answer the following questions about it.

1. What code do you need to get information from this input?
2. What's one example of when you would want to use this input?
3. What's an example of when you **wouldn't** want to use this input?

Teaching Tip

For this prompt, you can "jigsaw" the answers by assigning different students or different groups each a type of input and answering questions on each type. The different answers can then be combined for a class reference on input types.

Lesson 7: Analog Input

Overview

In this lesson, students explore how the three analog sensors (sound, light, and temperature) can be used to write programs that respond to changes in the environment. The use of these sensors marks a transition in terms of how users interact with a program. By using sensors as an input, the user of an app doesn't have to directly interact with it at all, or may interact without actually realizing they are doing so.

Purpose

This lesson builds on the previous by introducing analog sensors as a new form of input. These sensors all perform analog-to-digital conversion, allowing programs to sense things as represented by a 10 bit number (0-1023).

Agenda

Warm Up (10 min)
- **Analog and Digital**

Activity (40 min)
- **Analog Inputs**

Wrap Up (5 min)
- **Sensors All Around**

View on Code Studio

Objectives

Students will be able to:

- Develop programs that respond to analog input
- Scale a range of numbers to meet a specific need
- Represent a sensor value in a variety of ways

Preparation

- [ ] Cue up the **Difference between Analog and Digital Signals - Video**
- [ ] If your room isn't very bright, it's useful to have some flashlights or other light sources on hand for testing the light sensor

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Difference between Analog and Digital Signals** - Video

Vocabulary

- **Analog** - Any continuously changing signal that is not restricted to finite set of values. For example, the wave forms of spoken words are an analog signal.
- **Digital** - Data or signals represented by a finite number of values. Analog signals (which can have infinite values) must be converted to digital in order to be computed with.

Introduced Code

- `soundSensor`
- `soundSensor.value`
- `soundSensor.setScale(min, max)`
- `lightSensor`

- tempSensor

Teaching Guide

Warm Up (10 min)

Analog and Digital

Think, Pair, Share: How would you quantify each of your five senses (touch, taste, sight, smell, and hearing). Think back to the previous unit and the various ways you came up with to encode data for a computer.

Video: Today we're going to write programs that have "senses" of their own! Show **Difference between Analog and Digital Signals - Video**

Discussion Goal

The goal here is to get students thinking about how an natural (or analog) value can be converted into a computable format. Encourage students to consider what the upper and lower bounds of each value might be (for example, is there a minimum or maximum amount of sound?).

Activity (40 min)

Analog Inputs

Distribute: Pass out Circuit Playgrounds.

Transition: Send students to Code Studio. Let them know that today they will be experimenting with some sensors that detect analog signals and convert them to digital values that can be used by the computer.

Code Studio levels

Lesson Overview Student Overview

Sensor Experiment Student Overview

Analog Sensors Student Overview

Reading Sensors 4 5 6 7 *(click tabs to see student view)*

Polling Events Student Overview

Sensor Events 9 10 11 *(click tabs to see student view)*

Changing Sensor Scale Student Overview

Sensors to Colors 13 14 15 *(click tabs to see student view)*

Challenge: Making Sense of Sensors Student Overview

Wrap Up (5 min)

Sensors All Around

Journal: Considering all of the computing devices that you interact with on a regular basis, identify as many potential analog sensors as you can. Where do your computing devices take a continuously changing signal and convert it into digital data?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming
- ▶ **CS** - Computing Systems

Lesson 8: The Program Design Process

Overview

This lesson introduces students to the process they will use to design programs of their own throughout this unit. This process is centered around a project guide which asks students to sketch out their screens, identify elements of the Circuit Playground to be used, define variables, and describe events before they begin programming. This process is similar to the Game Design Process that we used in Unit 3. In this lesson students begin by playing a tug o' war style game where the code is hidden. They discuss what they think the board components, events, and variables would need to be to make the program. They are then given a completed project guide which shows one way to implement the project. Students are then walked through this process through a series of levels. At the end of the lesson students have an opportunity to make improvements to the program to make it their own.

Purpose

This lesson gives students to practice developing a larger scale program in order to prepare them to do so independently. While previous lessons have focused on building skills around using specific elements of the Circuit Playground and related programming concepts, this lesson focuses on combining everything learned so far into a more complex program. The lesson heavily scaffolds the software development process by providing students a completed project guide, providing starter code, and walking students through its implementation. In the subsequent lessons students will need to complete a greater portion of this guide independently, and for the final project they will follow this process largely independently.

Agenda

Warm Up (15 min)

- **Play Emoji Race**
- **Stop: Review Project Guide**

Activity (45-60 min)

- **Implement Project Guide**

Wrap Up (20 min)

- **Make It Your Own**

View on Code Studio

Objectives

Students will be able to:

- Implement different features of a program by following a structured project guide
- Develop a program that responds to events from a hardware input
- Create a function that uses parameters to generalize behavior

Preparation

- [ ] Provide students with copies of the **Emoji Race - Project Guide**

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Emoji Race** - Project Guide Make a Copy

Teaching Guide

Warm Up (15 min)

Play Emoji Race

Group: Place students in pairs

Demo: Show students the Emoji Race demo, available at the beginning of this lesson.

Transition: Send students online to try out the game on their own. Each pair should play the game and follow the instructions which ask them to list the board components, events, and variables they think are necessary to create this game.

Stop: Review Project Guide

Discuss: Students should have individually created a list of board components, events, and functions they would need to make the game they played. Ask students to share their lists with a neighbor before discussing as a class.

Distribute: Give each student or pair of students a copy of the **Emoji Race - Project Guide**

Prompt: Compare the list of things you thought would be needed to the ones on this project guide. Do you notice any differences?

Discuss: As a class compare the list you had on the board to the list of board components, events, and functions on the project guide. Note the similarities. Where there are differences try to understand why. Don't approach one set as "right" vs. "wrong" but just confirm both would be able to make the game students played.

Teaching Tip

Keeping Focus: Students can easily get distracted by the fun of playing the game. Let them play for a while but eventually encourage them to follow the on-screen instructions and make a list of the board components, events, and variables that would be necessary to create the game.

Discussion Goal

Running the Conversation: You can write "Board", "Events", and "Functions" on the board and record their ideas below each. Ask students to justify their decisions but don't feel the need to settle on one right answer.

Teaching Tip

Sharing the Project Guide: Students are not actually writing on the project guide in this lesson. You can give each student their own copy for reference but you might also choose to print one copy per pair, share digital copies, or just display the guide on the projector. So long as it is available for reference any approach will work fine.

Remarks

> There's usually lots of ways you can structure a program to get it to work the way you want. The important thing when writing complex or large programs is that you start with a plan. Today we're going to look at how we could implement this plan to build our own emoji race game. By the end of the lesson you'll not only have built your game, but you'll know how to change it and make it your own. Let's get going!

Activity (45-60 min)

Implement Project Guide

Students are given starter code to establish the functions and event handlers needed for this project. These levels guide students through how to use the provided variables and what ought to occur within the event handlers. As students move through the levels point out how the project guide is being used. Though this project is highly

scaffolded, the next lesson will ask students to develop a hardware-based program of their own design, so make sure to reinforce the connection between the planning that was done in the project guide and the programming levels.

Code Studio levels

Lesson Overview Student Overview

Sample Program Student Overview

Plan Your Project Student Overview

Screen Design 4 5 6 *(click tabs to see student view)*

Finishing Functions 7 8 9 *(click tabs to see student view)*

Checking for a Winner 10 11 12 13 *(click tabs to see student view)*

Make it Your Own Student Overview

Wrap Up (20 min)

Make It Your Own

This last level encourages students to make the game their own. If students have made their way to this point they have all the skills they need to progress through the curriculum, so there is no pressure to complete any of the modifications suggested in this level. If you have time, however, getting practice planning and implementing new features will be a useful skill. Even just modifying the look of the screens is an easy way students can make the game their own.

Share: Once students have completed the project they can share their work with their classmates. Encourage students to showcase the additional code they wrote and explain how it has changed the way the game works.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems

Lesson 9: Project: Make a Game

Overview

Students take what they've learned through chapter one, and develop an app of their own design that uses the board to output information.

Purpose

This end of chapter assessment is a good place for students to bring together all the pieces they have learned (event handlers, loops, using the board as output) in one place. This project is purposefully left very open to allow students thing broadly about how physical output might be useful in an app - this is a great opportunity to encourage students to revisit programs they've written earlier in this unit or in Unit 4 that would benefit from using the board to output information.

Agenda

Warm Up (10 min)

- **Demo Project Exemplars**

Activity (90 - 120 min)

- **Unplugged: Program Planning**
- **Prototyping the Program**

Wrap Up (10 min)

- **Sharing Projects**
- **Self Assessment**

View on Code Studio

Objectives

Students will be able to:

- Use event handlers to respond to user interaction
- Design a piece of software that uses hardware for non-traditional input and output
- Prototype a program that integrates software and hardware

Preparation

- [ ] Print a copy of **Make a Game - Project Guide** for each group of students
- [ ] Print a copy of **Make a Game - Rubric** for each student

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Make a Game** - Project Guide Make a Copy
- **Make a Game** - Rubric Make a Copy

Teaching Guide

Warm Up (10 min)

Demo Project Exemplars

Goal: Students see an example of a final project and discuss the different elements that went into making it.

Display: On the projector, show the example projects at the beginning of this lesson.

Prompt: Analyze the exemplar for the following elements:

- Where is input being taken?
  - What events do you think are being used for each input?
- What information is being output through the board?
- Where might this program be using a loop?

Discuss: Have students share their observations and analyses of the exemplar.

Distribute: Provide each student with a copy of the **Make a Game - Rubric**. Review the different components of the rubric with them to make sure they understand the components of the project.

Activity (90 - 120 min)

Unplugged: Program Planning

Group: Place students in groups of 2-3.

Distribute: Hand out the **Make a Game - Project Guide** to students. This is the tool students will use to scope out their projects before getting onto the computers. Give students some time to brainstorm the type of program they want to make.

Make a Game Project Guide

> **Brainstorming**
>
> Give groups some time to brainstorm ideas for their project. The goal is to come up with a simple game that can be controlled using elements of the Circuit Playground. Encourage students to think about all of the programs and projects that they've made so far as potential starting points (including the prototypes from Unit 4).
>
> 1. Once groups have settled on an idea, they can record it on the activity guide under Project Brainstorming.
> 2. Next students think through the Events they'll need to respond to.
> 3. Finally students consider the functions that they may need to create.

Teaching Tip

Students may find it particularly difficult to predict *all* of the events that they'll want to respond to or the functions that they'll need to create. Make sure that they know this is an *iterative* process and planning is only the first step. We want to start this project with as clear a plan as possible, but there will likely be things that weren't considered from the start and the plan will need to change accordingly.

Prototyping the Program

Transition: Once teams have completed their planning sheet, they can head to Code Studio to work on prototyping their projects.

Code Studio levels

Lesson Overview | **Student Overview**

Demo - Bug Grab Student Overview

Plan Your Project Student Overview

Screen Design 4 *(click tabs to see student view)*

Events 5 6 *(click tabs to see student view)*

Functions 7 8 *(click tabs to see student view)*

Finishing Touches 9 *(click tabs to see student view)*

Reflection 10 *(click tabs to see student view)*

Wrap Up (10 min)

Sharing Projects

Share: Find a way for groups to share their projects with each other. It will likely be helpful to use the share link for the project so that students can share the project with other students.

Self Assessment

Using the **Make a Game - Rubric**, students should assess their own project before submitting it.

Send students to Code Studio to complete their reflection on their attitudes toward computer science. Although their answers are anonymous, the aggregated data will be available to you once at least five students have completed the survey.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems

Lesson 10: Arrays and Color LEDs

Overview

An array is an ordered collection of items, usually of the same type. In this lesson, students learn ways to access either a specific or random value from a list using its index. They then learn how to access the colorLEDs array that controls the behavior of the color LEDs on the Circuit Playground. Students will control the color and intensity of each LED, then use what they have learned to program light patterns to create a light show on their Circuit Playground.

Purpose

Arrays are a common data structure in computer science, used to make storing and accessing information easier. The lesson prepares students to access items within an array, but doesn't yet cover creating and modifying arrays, which will be addressed in the following lesson. Students are also introduced to the ring of color LEDs, which are exposed as an array called `colorLeds`. Students learn how to access and control each LED in the array individually, preparing them to access multiple LEDs though iteration later in the chapter.

Agenda

Warm Up (5 min)

- **Discussion: What is a List?**

Activity (45 min)

- **Arrays and Color LEDs**

Wrap Up (5 min)

- **Discussion: Objects Vs Arrays**

View on Code Studio

Objectives

Students will be able to:

- Access an element in an array using its index
- Use the color LED array to individually control each color LED
- Use the color() and intensity() methods to control each color LED

Vocabulary

- **Array** - A data structure in JavaScript used to represent a list.

Introduced Code

- `colorLeds`
- `colorLeds[index].on()`
- `colorLeds[index].blink(interval)`
- `colorLeds[index].intensity(brightness)`
- `colorLeds[index].color(color)`
- `list[index];`

Teaching Guide

Warm Up (5 min)

Discussion: What is a List?

Set Up: Have students take out their journals. In their journals have students respond to the following prompt.

Prompt: What is a list? What do you use lists for?

Discuss: Once students have had time to write down their own thoughts, have them share out to the whole class. Work as a class to try to get an understanding of what a list is.

Discussion Goal

While the types of lists students come up with vary from class to class, the two main characteristics of lists that students should understand is that they usually are comprised of items of the same type (tasks to complete, names of people, song titles, grocery items, etc.) and that they have an order. Some common types of lists include a to-do list, a class roster, a playlist, or a grocery list.

Remarks

> When we program, we sometimes want to store lists of items. In Javascript, we do this using an array. An array is a collection of items that are stored in a particular order. We're going to look at different ways to use arrays, and then see how they will help us to control more LEDs on the Circuit Playground.

Activity (45 min)

Arrays and Color LEDs

Transition: Head to Code Studio

Code Studio levels

Lesson Overview — Student Overview

Introduction to Arrays — Student Overview

Introduction to Arrays — 3 | 4 | 5 | 6 | 7

(click tabs to see student view)

Using Arrays — Student Overview

Circuit Playground: Color LEDs — Student Overview

Color LEDs — 10 | 11 | 12 | 13 | 14 ***(click tabs to see student view)***

Light Show — 15 | 16 ***(click tabs to see student view)***

Wrap Up (5 min)

Discussion: Objects Vs Arrays

Prompt: You have now seen both objects, such as the Sprite object, and arrays. How is an array different from the attributes of an object? When would you use an array? When would you use an object?

You can either use this as an exit ticket or have a class discussion if you have time.

Content Corner

Arrays are frequently used to collect items of the same type (a bunch of numbers, a bunch of strings, a bunch of button names, etc), but in JavaScript all list items need not be of the same type. You can create lists the include numbers, strings, objects, and even other lists.

An Object is a data structure that combines many values (such as numbers, strings, and even functions) that represent the attributes of that object (height, width, rotation, name) which are easier to keep track of if done in one place. The attributes can be of many different types but they all describe one object.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems

Lesson 11: Making Music

Overview

In this lesson students will use the buzzer to its full extent by producing sounds, notes, and songs with the buzzer. Students start with a short review of the buzzer's frequency and duration parameters, then move on to the concept of notes. Notes allow students to constrain themselves to frequencies that are used in Western music and provide a layer of abstraction that helps them to understand which frequencies might sound good together. Onces students are able to play notes on the buzzer, they use arrays to hold and play sequences of notes, forming simple songs.

Purpose

This lesson allows students to get more creative with the buzzer by introducing the concept of a "note" and by using arrays to hold sequential collections of notes. Using notes rather than frequencies provides a layer of abstraction that makes it easier for students to identify sounds that can be used to make music. Arrays provide a way to group together sounds in sequence, so that they can be played as music. Using the `playNotes` block to iterate over an array of notes provides a conceptual foundation for working with `for` loops in the next lesson.

Agenda

Warm-Up (5 min)

- **Color LEDs vs the Buzzer**

Activity

- **Making Music**

Wrap-Up (5 min)

- **Journal**

View on Code Studio

Objectives

Students will be able to:

- Create and modify an array
- Use an array to produce sound on the buzzer
- Recognize an array as a list of elements that can be operated on sequentially.

Introduced Code

- `buzzer.note(note, duration)`
- `buzzer.playNotes(array, tempo)`
- `list.length`
- `var list = ["a","b","d"];`

Teaching Guide

Warm-Up (5 min)

Color LEDs vs the Buzzer

Prompt: We've been using the buzzer to make sounds, but those buzzes didn't always too great. What do you think you need to make real music on the buzzer?

Discussion Goal

Students may come up with many different ideas, but there are two aspects of this problem that should be highlighted before students move on to the activity. First, specifying frequencies is not the best way to indicate how high or low a buzz should be, since music only uses a limited number of frequencies, and other frequencies are considered "out of tune". Music is usually written down as specific notes on a scale, which makes it much easier to see which sounds will go well together. (Students with a musical background will likely have a much better grasp of this issue.) Second, students will need a way to string sounds, or notes, together to make a song. This second point should be more universally understood.

Activity

Making Music

Transition: Send students to Code Studio.

Code Studio levels

Lesson Overview Student Overview

Making Sounds 2 3 4 *(click tabs to see student view)*

Making Music Student Overview

Playing with Notes 6 *(click tabs to see student view)*

Modifying and Creating Arrays Student Overview

Musical Arrays 8 9 10 *(click tabs to see student view)*

Making Songs 11 12 13 14 *(click tabs to see student view)*

Wrap-Up (5 min)

Journal

Goal: Help students make the connection between the `playNotes` block and the for loops that they will use in the next lesson.

Prompt: Today, instead of just choosing one element from an array, we used the `playNotes` block to do something to every element. Think back to the other arrays you have seen. How might doing something to every element in an array be useful there?

Share After giving students time to reflect in their own journals, have them share with a partner or in small groups.

Lesson 12: Arrays and For Loops

Overview

Using a *for loop* to iterate over all of the elements in an array is a really useful construct in most programming languages. In this lesson, students learn the basics of how a *for loop* can be used to repeat code, and then combine it with what they've already learned about arrays to write programs that process all elements in an array. Students use for loops to go through each element in a list one at a time without having to write code for each element. Towards the end of the lesson students will apply this with the `colorLed` list on the board to create an app that changes all of the LEDs each time a button is clicked.

Purpose

As students start using arrays more frequently, a common pattern emerges wherein you want to run some code on each element of an array. While *for loops* are a generally useful structure for repeating code, they are a particularly useful for iterating over an array. In this lesson we build on the understanding of arrays that students have developed in the last two lessons by introducing the *for loop*, which combines a variable, the counter pattern, and a conditional all in a single construct.

Agenda

Warm Up (5 min)

- **Run Code on All Elements of an Array**

Activity (40 min)

- **Arrays and For Loops**

Wrap Up

- **Journal**

View on Code Studio

Objectives

Students will be able to:

- Modify the exit condition of a for loop to control how many times it repeats
- Use a for loop to iterate over an array

Vocabulary

- **Array** - A data structure in JavaScript used to represent a list.
- **For Loop** - Loops that have a predetermined beginning, end, and increment (step interval).

Introduced Code

- `for(var i=0; i<4; i++){ //code }`
- `function myFunction(n){ //code }`
- `Call a function with parameters`

Teaching Guide

Warm Up (5 min)

Run Code on All Elements of an Array

Discuss: If you had a list full of all of your ToDos and you wanted the computer to print out each one, how might you do it? Don't worry about the specific code, focus on the what information your program would need to know and keep track of. Would your approach still work if you added or removed a ToDo from your list?

Discussion Goal

You shouldn't expect students to know how *exactly* a computer would do this, but to start thinking about what is needed to keep track of where you are in an array (the index), how you would know you've reached the end of the array (the array length), and how you might incrementally increase where you are in the list (the counter pattern).

Remarks

> Doing something to every element of an array is a really common problem in Computer Science, and figuring out how to solve this problem will let us write more efficient programs. Today we're going to focus on three things to help us with this problem
>
> 1. Using the index to do something to elements in an array
> 2. Keeping track of a counter so we can move through the elements of an array by index
> 3. Using the length of an array to know when we've reached the end

Activity (40 min)

Arrays and For Loops

Transition: Send students to Code Studio

Code Studio levels

Lesson Overview | Student Overview

Programs that Repeat | 2 | 3 | *(click tabs to see student view)*

For Loops | Student Overview

CSD: For Loop | Student Overview

For Loops | 6 | 7 | 8 | 9 | *(click tabs to see student view)*

For Loops and Color LEDs | 10 | 11 | 12 | 13 | 14 | 15

(click tabs to see student view)

Wrap Up

Journal

Prompt: Have students reflect on their development of the **five practices of CS Discoveries** (Problem Solving, Persistence, Creativity, Collaboration, Communication). Choose one of the following prompts as you deem appropriate.

- Choose one of the five practices in which you believe you demonstrated growth in this lesson. Write something you did that exemplified this practice.
- Choose one practice you think you can continue to grow in. What's one thing you'd like to do better?
- Choose one practice you thought was especially important for the activity we completed today. What made it so important?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming
- ▶ **CS** - Computing Systems

Lesson 13: Accelerometer

Overview

In this lesson, students will explore the accelerometer and its capabilities. They'll become familiar with its events and properties, as well as create multiple programs utilizing the accelerometer similar to those they've likely come across in real world applications.

Purpose

This lesson gives students an opportunity to work with the accelerometer sensor and explore its orientation properties and accelerometer-specific board events. Students will see the purpose and uses of an accelerometer in real world devices and programs and create their own versions of some of these applications. To do this, students will need to refer back to their past knowledge of the counter pattern to create functional accelerometer-based apps.

Agenda

Warm Up (5 Min)

- **What Makes a Sensor?**

Activity

- **The Accelerometer**

Wrap Up (5 Min)

- **Journal**

View on Code Studio

Objectives

Students will be able to:

- Recognize the use and need for accelerometer orientation (pitch and roll).
- Identify and explain the difference between the shake, data and change events.
- Refer back to and use their past knowledge of the counter pattern.

Introduced Code

- `accelerometer.getOrientation()`

Teaching Guide

Warm Up (5 Min)

What Makes a Sensor?

Prompt: Refer back to the analog sensors, what makes a sensor a sensor? Could an accelerometer be a sensor?

Share: Have students share their thoughts and ideas in small groups.

Discussion Goal

Students have been exposed to sensors in previous lessons but have only seen a few aspects of what a sensor can measure. This discussion gives students a chance to think about the characteristics of a sensor, while also thinking about the other possible characteristics of a sensor they haven't been exposed to.

Activity

The Accelerometer

Transition: Send students to Code Studio.

Code Studio levels

Lesson Overview Student Overview

The Accelerometer Student Overview

Orientation 3 4 5 6 *(click tabs to see student view)*

Using Accelerometer Events Student Overview

Accelerometer Events 8 9 *(click tabs to see student view)*

Revisiting the Counter Pattern Student Overview

Accelerometer Events 11 12 13 *(click tabs to see student view)*

Wrap Up (5 Min)

Journal

Goal: Students should be able to recognize the use for an accelerometer.

Prompt: Now that you've created your own programs with the accelerometer, can you think of some ways you interact with accelerometers every day through technology?

Lesson 14: Functions with Parameters

Overview

The lesson starts with a quick review of parameters, in the context of App Lab blocks that they students have seen recently. Students then look at examples of parameters within user-created functions in App Lab and create and call functions with parameters for themselves, using them to control multiple elements on a screen. Afterwards, students use for loops to iterate over an array, passing each element into a function. Last, students use what they have learned to create a star catching game.

Purpose

In previous lessons, students have used functions to define blocks of code that can be used in multiple places in a program. In this lesson, students learn how to use parameters to generalize the purpose of a function. Parameters allow a program to specify the details of how a function works when it is called, rather than when the program is defined. Although students have seen functions with parameters earlier in the unit, this is the first time that they are expected to define and call their own. Students also learn how to use a for loop to iteratively pass in the elements of an array as parameters to a function, allowing them to use the same function on multiple elements on the screen.

Agenda

Warm Up
Activity
Wrap Up

View on Code Studio

Objectives

Students will be able to:

- Use parameters to generalize the purpose of a function.

Vocabulary

- **Parameter** - An extra piece of information passed to a function to customize it for a specific need

Introduced Code

- `function myFunction(n){ //code }`
- `Call a function with parameters`

Teaching Guide

Warm Up

Prompt: When you needed to access the pitch and roll of the accelerometer in the last lesson, you used the block `accelerometer.getOrientation`, and you chose whether to get the pitch or the roll. Why do you think the creators make the program work that way, rather than having two blocks, one for the pitch and one for the roll?

What other blocks that you have seen take parameters? Why are parameters so useful?

Discussion Goal

Goal: Students should eventually see that parameters give a program flexibility. Sometimes, multiple commands are similar enough that it makes sense to combine them into one, but with a parameter to distinguish between their differences. Sometimes, such as the case of `buzzer.frequency`, there are too make options to make a separate block for each one. Parameters allow a programmer to use a single solution to solve multiple, related problems.

Remarks

> So far, the functions that you have used always did the exact same thing. Today, we're going to look at a way to make functions even more useful by giving them parameters, just like some of the blocks that you just talked about.

Activity

Send students to Code Studio

Code Studio levels

Lesson Overview | Student Overview

Functions with Parameters | 2 | 3 | 4 | 5

(click tabs to see student view)

Iteration and Parameters | 6 | 7 | 8 ***(click tabs to see student view)***

Star Chaser | 9 | 10 | 11 | 12 | 13 | 14

(click tabs to see student view)

Wrap Up

Prompt: Think back to some of the programs that you have made before. What are two times that you could have used functions with parameters? What would the parameter be? How should the function's behavior change when the parameter changes?

If you are interested in licensing Code.org materials for commercial purposes, **contact us**.

Lesson 15: Circuits and Physical Prototypes

Overview

In preparation for this chapter's final project, students will learn how to develop a prototype of a physical object that includes a Circuit Playground. Using a modelled project planning guide, students will learn how to wire a couple of simple circuits and to build prototypes that can communicate the intended design of a product, using cheap and easily found materials such as cardboard and duct tape.

Purpose

The goal of this lesson is both to model for students how thinking about the physical design of a product impacts the prototyping process, and to introduce a handful of practical skills that will make creating their final projects easier.

Agenda

Warm Up (5 min)
- **Designing a Physical Device**

Activity 1 (30 min)
- **Introducing the Smart Bike**

Activity 2 (30-45 min)
- **Building the Prototype**

Activity 3 (30-45 min)
- **Adding Inputs**

Wrap Up (15 min)
- **Sharing Designs**

View on Code Studio

Objectives

Students will be able to:

- Create and debug simple circuits
- Develop an interactive physical protype that combines software and hardware
- Consider the needs of diverse users when designing a product

Preparation

- [ ] Gather prototyping materials, such as:
  - [ ] Structural material (cardboard, construction paper, etc)
  - [ ] Connective material (tape, glue, hot glue, etc)
  - [ ] Construction tools (scissors, staplers, etc)
  - [ ] Other materials (cups, binder clips, paper plates, etc)
- [ ] Prepare circuit wiring materials, such as:
  - [ ] Alligator clip wires (included in Circuit Playground classroom kit)
  - [ ] LEDs (included in Circuit Playground classroom kit)
  - [ ] Other conductive material (wire, paper clips, foil, etc)
  - [ ] (optional) Buttons or switches
- [ ] Print a copy of **Physical Prototyping - Project Guide** for each group of 2-3 students
- [ ] Prepare a model button to show the class

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **How Computers Work: Circuits & Logic** - Video (**download**)
- **Physical Prototyping** - Project Guide Make a Copy

Vocabulary

- **Circuit** - A device that provides a path for an electric current to flow, often modifying that current. In computers, circuits allow for simple logical and mathematical operations using electricity.
- **Prototype** - A first or early model of a product that allows you to test assumptions before developing a final version.

Introduced Code

- `pinMode(pin, mode);`
- `digitalWrite()`
- `var myLed = createLed(pin);`
- `var myButton = createButton(pin);`

Teaching Guide

Warm Up (5 min)

Designing a Physical Device

Prompt: If you could create any kind of computational device, what would it be? What would it do? How would people interact with it?

Share: After a few minutes of thinking time, have students share their ideas.

Discuss: How might a device like the Circuit Playground help us design and prototype some of these ideas? Consider picking a couple of ideas to put up on the board and list with the class which things features of a given device could be replicated with an element of the board, and which might require additional hardware.

Discussion Goal

The point of this short discussion is to get students thinking about how we might use a board like the Circuit Playground as part of a larger computing device. It's not important that we're able to completely replicate student's ideas with the board, but that we can start thinking about where the board could be used and where we might need to add additional functionality.

Activity 1 (30 min)

Introducing the Smart Bike

Group: Place students into groups of 3-4.

Distribute: Give each group a copy of **Physical Prototyping - Project Guide**, and introduce the project. Give students a moment to look over the activity guide. This guide is similar to the one they will be using for their own final projects, but it's already been mostly completed, which allows for them to focus on how to implement the idea, instead of what the idea should be.

Discuss: Focusing on the description and sketch on the first page, how might we use the Circuit Playground to develop this prototype. Which elements are we currently *unable* to replicate with the board?

Display: Watch the **How Computers Work: Circuits & Logic - Video** (level two in the progression) as a class.

Remarks

> The elements of our Circuit Playground are all made up of circuits so small that we can't even see most of them, but you can create a simple circuit on your own by attaching wires to the copper pads on the edge of the board.

Discussion Goal

Students should be able to identify that almost all of the functionality (such as blinking lights, responding to button presses, and buzzing) can be done with the Circuit Playground as is. The major barrier we want to identify is how the physical requirements of the plan (such as placing the blinker lights on the ends of the handlebars) would require adding new hardware to the board. This is intended to tee up the next activity, in which students wire additional circuits onto their boards.

Distribute: Wires, LEDs, and any other hardware you want available for this portion. Let the class know that before digging too far into building our prototypes, we'll need to learn how to add additional hardware to the board.

Transition: Head to Code Studio to work on the Simple Circuits section.

Code Studio levels

- Levels

- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Student Instructions

View on Code Studio

Make a Prediction

All of the devices that you've used so far are actually circuits connected to numbered pins! Look for **#13** on your board to see which circuit is connected to pin 13, then read through this code and predict what will happen when the program is run.

Student Instructions

View on Code Studio

Student Instructions View on Code Studio

Wire a Circuit

You can use any of the numbered pads to add additional circuits to your board. Let's use pin 2 to add another LED.

Do This

- Using a wire, connect pin 2 to the positive (+) side of an LED
- Using another wire, connect the negative (-) side of the LED to a ground (GND) pin (it doesn't matter which one)
- Run this program to test your circuit. If it worked, your LED should turn on.

Debugging Tip: LEDs only work if the electricity is flowing from positive to negative. If your LED doesn't light up, make sure that it's oriented the right way.

Student Instructions View on Code Studio

Teaching Tip

Learning More About LEDs

We include LED sequins in the Circuit Playground classroom pack, but if you're careful you can use just about any LEDs you like. To learn more about pairing LEDs with the proper resistor, check out **Adafruit's LED primer**.

Teaching Tip

Why Are We Talking Pins?

You may notice that in the early levels of this lesson students are programming in a much different way than they have before, directly manipulating pins. The `digitalWrite()` and `pinMode()` commands are useful to get our wiring set up quickly, but are not the core concerns of this lesson. As soon as students are comfortable wiring LED circuits we will introduce techniques to control those LEDs with the higher level commands that have been used up to this point (like `led.on()` and `led.off()`).

Creating Board Objects

You might have noticed that the new blocks we're using are in a different toolbox drawer. The **Circuit** drawer contains all of the board objects that are built into the Circuit Playground, but when you start wiring your own circuits the Maker Toolkit no longer knows where everything is.

The new **Maker** drawer contains general purpose commands instead of ones that are customized for the Circuit Playground. In addition to the `pinMode()` and `digitalWrite()` commands you've seen, it includes commands to create new objects on the board that can be programmed in the same way as the blocks in the **Circuit** drawer. The `var myLed = createLed()` command, for example, creates a new LED object that behaves just like the `led` blocks you've been using.

Do This

Now that you're creating new board objects that we don't have blocks for, you'll need to work in text mode. You can still drag out blocks that you're familiar with from the **Circuit** drawer, you'll just need to change the name of the object.

- Keep your LED wired just as it was before (connected to pin 2)
- Make sure you're in text mode, not block mode.
- Drag out an `led.blink()` block below the comment `Blink myLed`.
- Replace the text `led` with `myLed`.
- Test your code.

Student Instructions **View on Code Studio**

Wiring Multiple LEDs

Using the `createLed()` block you can connect and control as many LEDs as your board has room for. Each LED needs to be connected to a separate numbered pin, but they can all share the same ground pin.

Do This

Leave the current LED connected to pin 2, but add another one to a numbered pin of your choice. For your new LED:

- Add a `var myLed = createLed()` block.
- Replace the variable label `myLed` with a unique label.
- Make sure you're in text mode, not block mode.
- Add an `led.blink()` command.
- Replace the text `led` with your new LED variable.
- Test your code.

Challenge: Try adding a third LED and make all three LED blink at different intervals.

Student Instructions **View on Code Studio**

Smart Bike - Blinkers

Using your planning guide, wire up the two LEDs that will serve as the blinkers. In order to make sure that the blinkers can be mounted at the end of the handlebars, make sure you wire them so that they can stretch out in opposite directions.

Do This

Using alligator clips, wire, or other conductive material, connect two LEDs to your board, one for each turn signal blinker.

- Hook up each LED to a different numbered pin on the board.
- Use the `createLed()` block to create an LED object for each blinker.

Tip: Make your wiring easier by considering how your blinkers will be mounted when selecting a pin to use.

Student Instructions **View on Code Studio**

Smart Bike - Blinker Controls

With your turn signal LEDs hooked up, you just need to program some buttons to control them. You may want to place buttons elsewhere on the bike to make controlling your turn signals easier, but for now we'll just use the built in left and right buttons.

Do This

- Add event handlers to blink the left turn signal when the left button is pressed.
- Add event handlers to blink the right turn signal when the right button is pressed.
- Test your code!

Hint: You'll need to be in text mode to make the blinkers work, since there are no built-in blocks for the elements that you add on to the board.

Student Instructions

View on Code Studio

Build Your Blinkers

Using the circuit you just built, return to your smart bike prototype and add the blinkers.

Activity 2 (30-45 min)

Building the Prototype

Transition: At this point students should have put together the two additional LED circuits necessary for their turn signals. Now we're going to transition off of the computers for a bit to work on building that circuit into a physical prototype.

Distribute: Make available any remaining building materials that you've gathered, such as cardboard and tape.

Teaching Tip

Sharing Boards? If you are sharing boards among multiple classes, you'll need to take that into consideration before groups start to build their projects. You may want to add a design constraint that the board must be easily removable from the design, or to create group sizes large enough to ensure that each group in every class has access to a board at all times.

Build: Allow groups some time to build their prototypes, using the board and LED circuits. While there's a lot of potential functionality that we could build at this point based on the guide, make sure that students are focusing on making the turn signals work.

Teaching Tip

To provide more student ownership and creativity, consider allowing students to modify the form (but not necessary functionality) of the smart bike controller. For example, they may want to turn it into something that the rider could wear, or something that works on skateboards or other modes of transportation. The import element here is that we work in a form that requires adding additional circuits to the board, instead of continuing to work solely within the contrainsts of the board's form.

Test: Once the basic prototype is built, have students test the code they wrote. It's not uncommon at this point to see new bugs that weren't present when testing the software alone. Encourage students to test all elements of their new circuits, ensuring that all of the connection points are solid, that the circuits are connected to the correct pins, and that the LEDs are oriented correctly.

Activity 3 (30-45 min)

Adding Inputs

Display: Put up the sketch on the first page of **Physical Prototyping - Project Guide** where the whole class can see it. The first prototype that we've constructed takes care of the turn signals, but now we're going to focus on the "horn" feature. Clearly we can use the buzzer to make the horn sound, but how can we control the horn?

Transition: Send students back to the online progression to learn about creating button circuits, which they will then add to their smart bike prototypes. This portion of the progression will take them all the way through finalizing their designs and submitting the code.

Code Studio levels

- Levels
- **11**
- **12**
- **13**
- **14**
- **15**

Student Instructions

View on Code Studio

Student Instructions

View on Code Studio

Make Your Own Buttons

Similar to LEDs, buttons are a really simple circuit that you can add to your board pretty easily. Like LEDs, buttons should be wired from a numbered pin to a ground pin, but unlike LEDs, a button circuit should be disconnected in the middle. When you connect the circuit, it will produce a button press event.

Do This

- Grab two wires.
- Connect one wire to an open numbered pin.
- Connect the second wire to a ground pin.
- Update line 2 so that it's referencing the pin you chose.
- Run the provided code.
- With the program running, touch the unconnected ends of both wires together to "press" the button.

Tip: A button circuit can be made with many different kinds of materials, as long as they are electrically conductive. Try making buttons with foil, silverware, or paper clips.

Student Instructions

Smart Bike - Buzzer

Now that we know how to add more buttons, you can add a button to control the smart bike's horn.

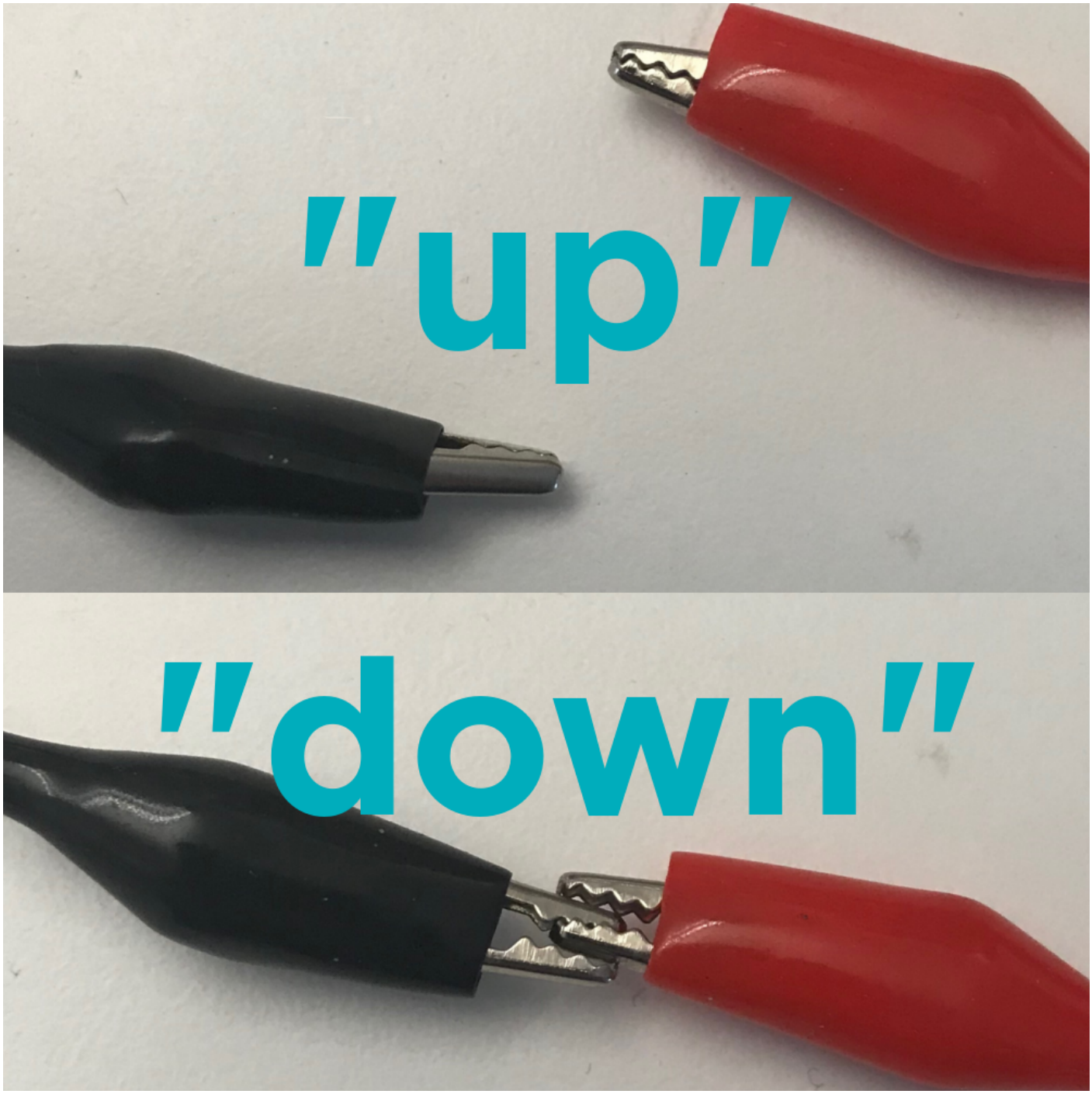
Do This

Using the button that you've already wired to the board, find a good spot to place your horn button. Then:

- Create a button object for your horn button
- Add an event handler to buzz when button is pressed

Hint: The button object that you create won't be in the `onBoardEvent()` *dropdown, so you'll need to type the name you've chosen in. Make sure not to use quotation marks!*

Student Instructions



Smart Bike - Headlight

View on Code Studio View on Code Studio

The last part of the smart bike plan that we need to figure out is the automatic headlight.

Do This

First you'll need to decide what to use for your headlight. You could add another LED circuit, or perhaps you can find a way to mount the board that allows for using the color LEDs as a headlight. Once you've figured out the physical layout of your lights, add code to your program that turns on and off the headlight based on how light or dark it is.

Hint: If you're using the built-in color LEDs, you might need to protect the light sensor to make sure that it's responding to the ambient light level and not the light from the LEDs.

Student Instructions

View on Code Studio

Smart Bike - Final Touches

At this point your smart bike should have all of its basic functionality in place. Now is your chance to add any finishing touches.

Do This

You may want to divide and conquer at this point, allowing some members of your group to focus on the physical aspects of the prototype while others work on improving the code. As this is a prototype, don't worry about making everything perfect, but do try to ensure that the prototype communicates your design well enough to test and get feedback.

Wrap Up (15 min)

Sharing Designs

Share: Give each group an opportunity to share their designs. Take some time to notice and celebrate the differences in design - even in a situation where we are all working from the same plan, each individual will bring their own experiences and perspectives to the design of a product.

Discuss: Using some of the specific design choices as an example, discuss with the class how their design choices impact the usability of their products. What assumptions did we make about our users, and how might our design choices have excluded or disadvantaged different user groups.

Discussion Goal

This discussion is intended to tie the physical design work students are doing now to the work they did in unit 4. Though we haven't explicitly asked students to spend the same amount of energy thinking about users and their needs in this project, it's still an essential element of good design. Push your students to consider how the design of *physical* computing devices brings new challenges and opportunities when it comes to designing for a diverse set of users.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems
- **IC** - Impacts of Computing

Lesson 16: Project: Prototype an Innovation

Overview

In this final project for the course, students team to develop and test a prototype for an innovative computing device based on the Circuit Playground. Using the inputs and outputs available on the board, groups will create programs that allow for interesting and unique user interactions.

Purpose

This lesson is the culmination of Unit 6 and provides students an opportunity to build a Maker Toolkit project of their own from the ground up. This project is an opportunity to showcase technical skills, but they will also need to demonstrate collaboration, constructive peer feedback, and iterative problem solving as they encounter obstacles along the way. This project should be student-directed whenever possible, and provide an empowering and memorable conclusion to the final unit of CS Discoveries.

Agenda

Warm Up (10 min)

- **Review Project Guide**

Activity (80-200 min)

- **Define - Scope Innovation**
- **Prepare - Complete Project Guide**
- **Try - Develop Prototypes**
- **Reflect - Peer Review**
- **Iterate - Revise Prototypes**

Wrap Up (30-60 min)

- **Share**

Extensions

- **Pitch Video**
- **Marketing Website**
- **Crowdfunding Campaign**

View on Code Studio

Objectives

Students will be able to:

- Independently scope the features of a piece of software
- Prototype a physical computing device
- Implement a plan for developing a piece of software that integrates hardware inputs and outputs

Preparation

- [ ] Collect materials for physical prototyping, eg.
  - [ ] Cardboard
  - [ ] Scissors
  - [ ] Tape
  - [ ] Glue
  - [ ] Foil
- [ ] Print a copy of **Prototype an Innovation - Project Guide** for each pair of students
- [ ] Print a copy of **Prototype An Innovation - Peer Review** for each student
- [ ] Print a copy of **Prototype An Innovation - Rubric** for each student

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Prototype an Innovation** - Project Guide Make a Copy
- **Prototype An Innovation** - Peer Review Make a Copy
- **Prototype An Innovation** - Rubric Make a Copy

Teaching Guide

Warm Up (10 min)

Review Project Guide

Group: Place students into groups of 2-4 for this project. At your discretion you may choose to have students form larger groups or work independently.

Distribute: Provide a copy of **Prototype an Innovation - Project Guide** to each group. As a class, review the different steps of the project and where they appear in the project guide. Direct students towards the **Prototype An Innovation - Rubric** so that they know from the beginning what components of the project you will be looking for.

Activity (80-200 min)

Define - Scope Innovation

Brainstorm: Students should spend the first 15-20 minutes brainstorming ideas for innovative devices built around the features of the Circuit Playground. Encourage students to review their prior work in this unit as well as the real world innovations they researched earlier for inspiration.

Prepare - Complete Project Guide

Distribute: Make available any construction materials that students may need for building their prototypes. While the focus of this step isn't to build the actual prototype, having these materials available can help with the brainstorming process.

Circulate: Once students have discussed their ideas for the project they should complete the **Prototype an Innovation - Project Guide**. While this should be a fairly familiar process, encourage students to make each component as clear and detailed as they can at this point. Planning ahead can help them identify issues in their plan before they'll need to make more significant changes to their code or physical device. Encourage students to use additional paper to sketch out screens for their app or additional views of their physical device.

Try - Develop Prototypes

Distribute: Make available the physical prototyping materials (such as cardboard, tape, scissors, etc). Let students know that, just as we used paper prototypes to quickly test software ideas, hardware developers often used cheap materials such as cardboard and tape to quickly iterate on the design of physical devices. While not all student ideas may require a physically prototyped component, you should encourage students to consider how the shape and design (or form factor) or their innovation could impact its usability.

Transition: Depending on the nature of their innovations, students may need to spend some time building the physical components of their projects before moving online. When students are ready to program, they can transition to Code Studio. These levels provide some guidance on how students may go about implementing their projects, but are left quite open to allow for a broad range of ideas. If they wish, students can work in a different order than the one suggested in these levels.

Code Studio levels

Lesson Overview | **Student Overview**

Design Your Prototype | **Student Overview**

User Interface 3 4 *(click tabs to see student view)*

Board Input and Output 5 6 *(click tabs to see student view)*

Finishing Touches 7 8 *(click tabs to see student view)*

Reflection 9 *(click tabs to see student view)*

Reflect - Peer Review

Group: Pair up groups to review each other's projects.

Distribute: Give each student a copy of **Prototype An Innovation - Peer Review**. Students should spend 15 minutes reviewing the other group's project and filling out the peer review guide.

Teaching Tip

If you have the time for it, this can be a good opportunity to pull in the user testing process that students learned in Unit 4.

Iterate - Revise Prototypes

Circulate: Students should complete the peer review guide's back side where they decide how to respond to the feedback they were given. They should then use that feedback to improve their game.

Wrap Up (30-60 min)

Share

Share: Give students a chance to share their innovations, either within the class or to a broader audience. If you choose to let students do a more formal presentation of their projects the project guide provides students a set of components to include in their presentations including:

Teaching Tip

Celebrate: As this is the culminating project for the entire course, consider going big with this share out. Bring in parents and administrators, or even host a after school event to provide students with a real audience to share their accomplishments with.

- The original innovation they set out to build
- A description of the programming process including at least one challenge they faced and one new feature they decided to add
- A description of the most interesting or complex piece of code they wrote
- A live demonstration of the actual innovation

Send students to Code Studio to complete their reflection on their attitudes toward computer science. Although their answers are anonymous, the aggregated data will be available to you once at least five students have completed the survey.

Extensions

Pitch Video

Record and edit a short video to pitch your innovation.

Marketing Website

Using Web Lab, design a website to market your innovation.

Crowdfunding Campaign

Have students design a crowdfunding campaign (along the lines of Kickstarter or Indiegogo) for their innovations. This could include:

- Design mockups for the final product
- A rough cost analysis for production
- A short pitch video

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming
- ▶ **CS** - Computing Systems