Variables

This lesson introduces the use of variables to store data or the results of mathematical operations. Students will practice giving variables unique and meaningful names. We will also introduce the basic mathematical operations for adding, subtracting, multiplying, and dividing variables.

Lesson Objectives

Students will...

- Understand what variables are and why and when to use them in a program.
- Learn how to create a variable, set the variable to an initial value, and change the value of the variable within a micro:bit program.
- Learn how to create meaningful and understandable variable names.
- Understand that a variable holds one value at a time.
- Understand that when you update or change the value held by a variable, the new value replaces the previous value.
- Learn how to use the basic mathematical blocks for adding, subtracting, multiplying, and dividing variables.
- Apply the above knowledge and skills to create a unique program that uses variables as an integral part of the program.

Lesson plan

1. Introduction

Computer programs process information. Some of the information that is input, stored, and used in a computer program has a value that is constant, meaning it does not change throughout the course of the program. An example of a constant in math is ‘pi’

because ‘pi’ has one value that never changes. Other pieces of information have values that vary or change during the running of a program. Programmers create variables to hold the value of information that may change. In a game program, a variable may be created to hold the player’s current score, since that value would change (hopefully!) during the course of the game.

Ask the students to think of some pieces of information in their daily life that are constants and others that are variables.

> What pieces of information have values that don’t change during the course of a single day (constants)?
> What pieces of information have values that do change during the course of a single day (variables) Constants and variables can be numbers and/or text.

Examples

In one school day…

> Constants: The day of the week, the year, student’s name, the school’s address
> Variables: The temperature/weather, the current time, the current class, whether they are standing or sitting…

Variables hold a specific type of information. The micro:bit’s variables can keep track of numbers, strings, booleans, and sprites. The first time you use a variable, its type is assigned to match whatever it is holding. From that point forward, you can only change the value of that variable to another value of that same type.

> A number variable could hold numerical data such as the year, the temperature, or the degree of acceleration.

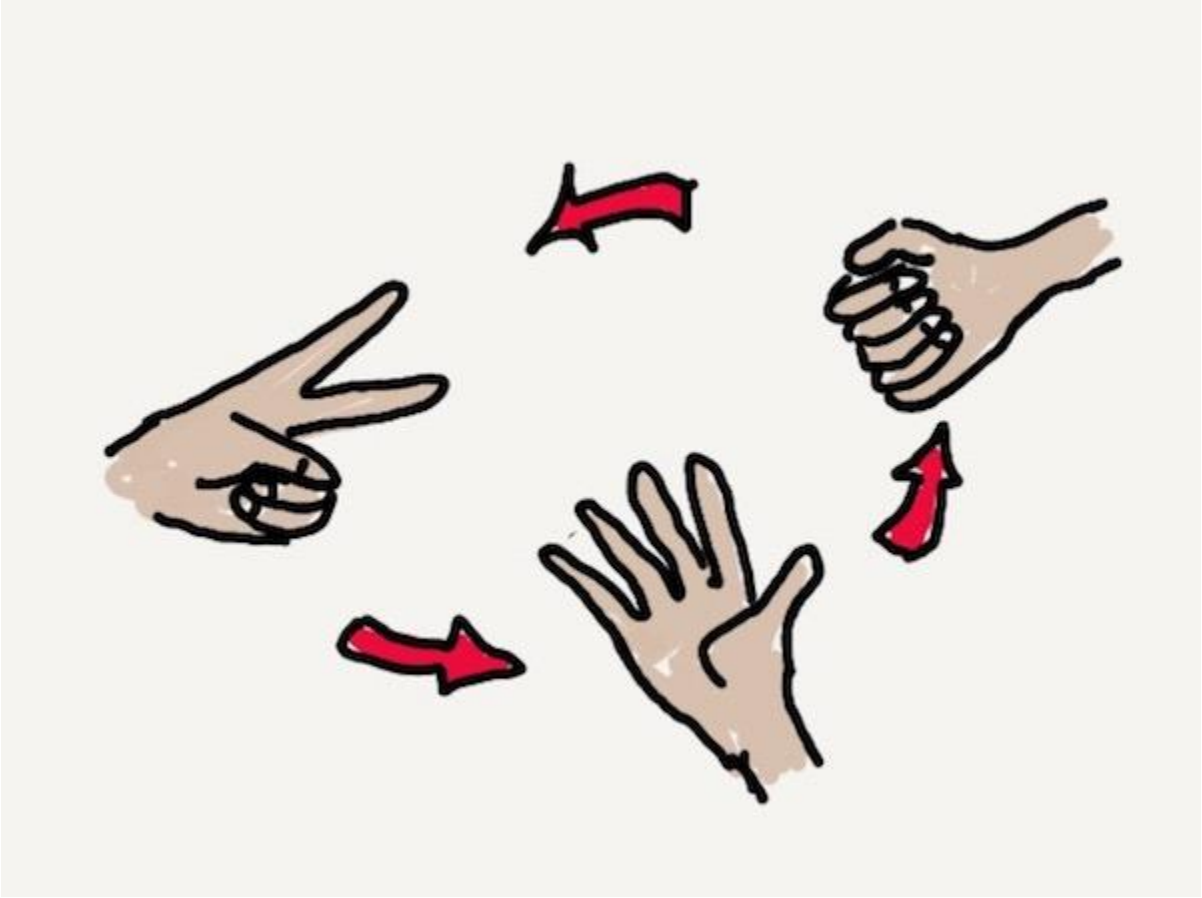
A string variable holds a string of alphanumeric characters such as a person's name, a password, or the day of the week.

A boolean variable has only two values: true or false. You might have certain things that happen only when the variable called *gameOver* is false, for example.

A sprite is a special variable that represents a single dot on the screen and holds two separate values for the row and column the dot is currently in.

2. Unplugged: Keeping score

The objective of this activity is to experience creating and working with variables by pairing up and playing *Rock Paper Scissors*.



Ask students to keep track of their scores on paper. You can also have students play in groups of three with the third student acting as the scorekeeper.

Students will keep track of how many times each player wins as well as the number of times the players tie.

Play: Have students play Rock Paper Scissors for about a minute. When done, ask the students to add up their scores and how many ‘rounds’ they played.

Play again: Tell students they will now start over and play again for another minute. When done, ask the students to add up their scores and how many ‘rounds’ they played.

Ask some students to share how they kept track of player scores. There may be some variety, but most will have written down the players’ names and then beside or below the names, marks representing the ‘wins’ of each player. And they may have made a separate place for recording ties.

Ask the students what parts of the score sheet represent constants, values that do not change through the course of a gaming session.

Example: The players’ names are constants.

Ask the students what parts of the score sheet represent variables, values that do change through the course of a gaming session.

Example: The players’ number of wins are variables.

3. Activity: Scorekeeper

This micro:bit activity guides the students to create a program with three variables that will keep score for their *Rock Paper Scissors* game.

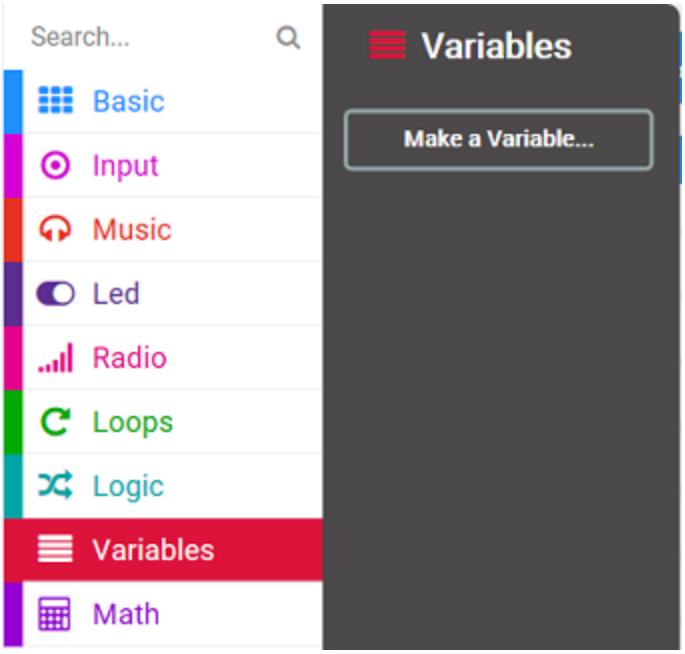
Tell the students that they will be creating a program that will act as a scorekeeper for their next Rock Paper Scissors game. They will need to create variables for the parts of scorekeeping that change over the course of a gaming session. What are those variables?

> The number of times the first player wins
> The number of times the second player wins
> the number of times the players tie

Creating and naming variables: Lead the students to create meaningful names for their variables.

> What would be a unique and clear name for the variable that will keep track of the number of times Player A wins?
> Student suggestions may be: `PAW`, `PlayerA`, `AButtonPress`, `AButtonCount`, `PlayerAWins`...
> Discuss why (or why not) different suggestions make clear what value the variable will hold. In general, variable names should clearly describe what type of information they hold.

In MakeCode, from the Variables menu, make and name these three variables: `PlayerAWins`, `PlayerBWins`, `PlayersTie`.



Initializing the variable value

It is important to give your variables an initial value. The initial value is the value the variable will hold each time the program starts. For our counter program, we will give each variable the value 0 (zero) at the start of the program.

Updating the variable value

In our program, we want to keep track of the number of times each player wins and the number of times they tie. We can use the buttons A and B to do this.

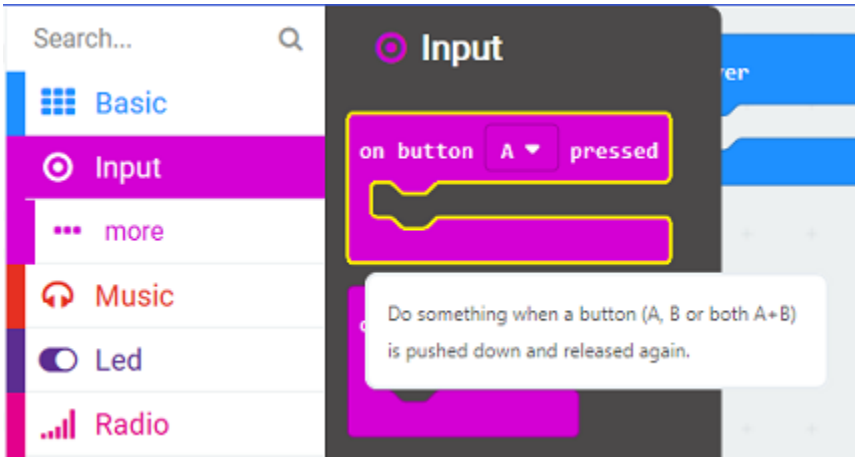
Pseudocode:

> Press button A to record a win for player A
> Press button B to record a win for player B
> Press both button A and button B together to record a tie

We already initialized these variables and now need to code to update the values at each round of the game.

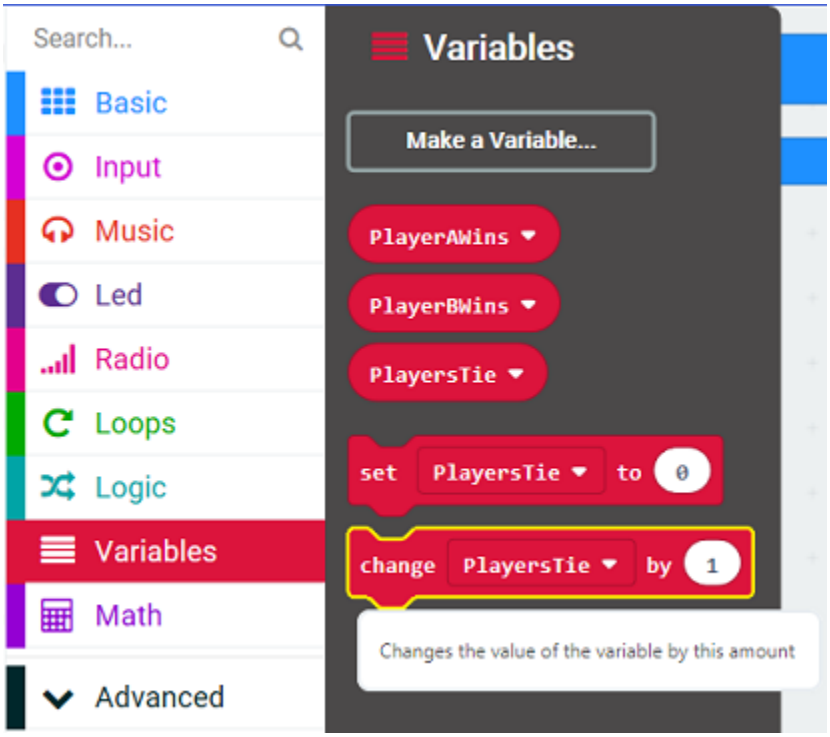
> Each time the scorekeeper presses button A to record a win for Player A, we want to add 1 to the current value of the variable `PlayerAWins`.
> Each time the scorekeeper presses button B, to record a win for Player B, we want to add 1 to the current value of the variable `PlayerBWins`.
> Each time the scorekeeper presses both button A and button B at the same time to record a tie, we want to add 1 to the current value of the variable `PlayersTie`.

From the Input menu, drag 3 of the 'on button A pressed' event handlers to your Programming Workspace.



Leave one block with ‘A’. Use the drop-down menu in the block to choose ‘B’ for the second block and ‘A+B’ for the third block.

From the Variables menu, drag 3 of the ‘change PlayersTie by 1’ blocks to your Programming Workspace.



Place one change block into each of the Button Pressed blocks.

Choose the appropriate variable from the pull down menus in the change blocks.

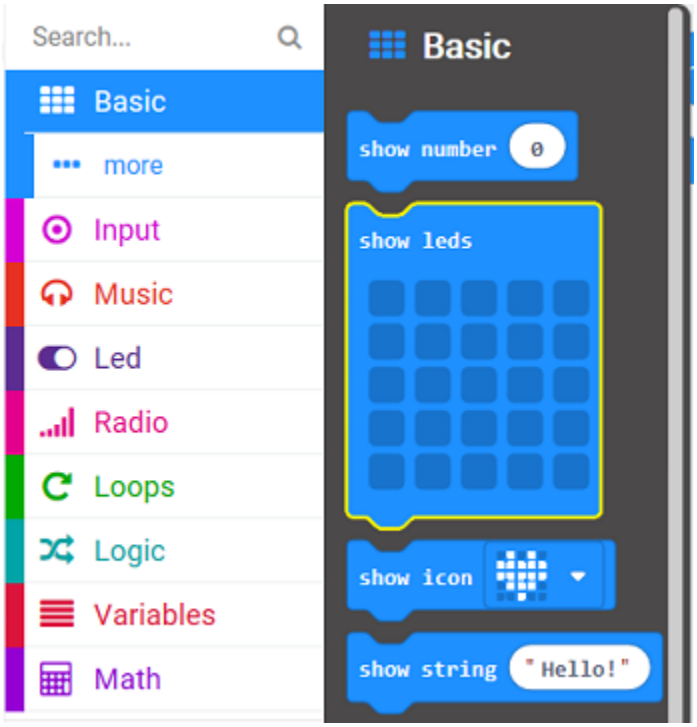
User feedback

Whenever the scorekeeper presses button A, button B, or both buttons together, we will give the user visual feedback acknowledging that the user pressed a button. We can do this by coding our program to display:

an ‘A’ each time the user presses button A to record a win for Player A,
a ‘B’ for each time the user presses button ‘B’ to record a win for Player B,

a ‘T’ for each time the user presses both button A and button B together to record a tie.

We can display an ‘A’, ‘B’, or ‘T’ using either the ‘show leds’ block or the ‘show string’ block.



In this example, we have used the ‘show leds’ block.

Notice that we added a ‘clear screen’ block after showing ‘A’, ‘B’, or ‘T’. What do you think would happen if we did not clear the screen? Try it.

Showing the final values of the variables

To finish our program, we can add code that tells the micro:bit to display the final values of our variables. Since we have already used buttons A and B, we can use the ‘on shake’ event handler block to trigger this event. We can use the ‘show string’, ‘show

leds', 'pause', and 'show number' blocks to display these final values in a clear way. Here is the complete program.

Buttons have been used as human input devices since computers first existed. Watch this video and see how they let tell the micro:bit to do something.

Try it out!

Download the Scorekeeper program to the micro:bit, and have the students play one last round of Rock Paper Scissors using their micro:bits to act as the Scorekeeper!

'Adding' on with mathematical operations

There is more we can do with the input we received using this program. We can use mathematical operations on our variables.

Example: Perhaps you'd like to keep track of, and show the player the total number of 'rounds' that were played. To do this, we can add the values stored in the variables we created to keep track of how many times each player won and how many times they tied.

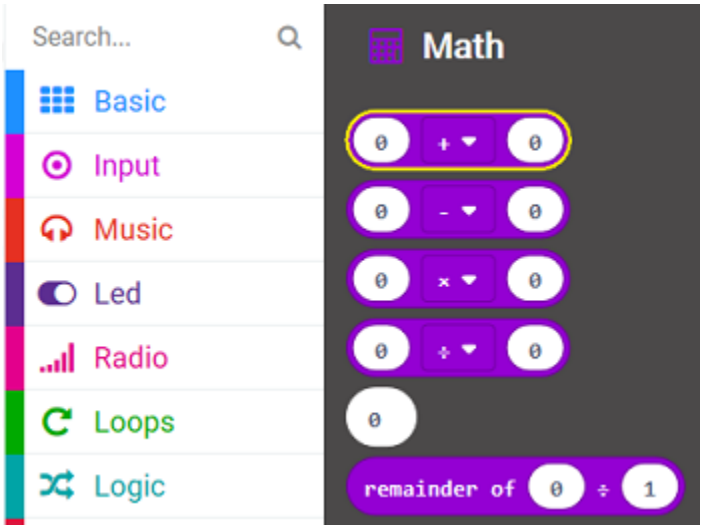
In order to do this, we can add the code to our program under the 'on shake' event handler.

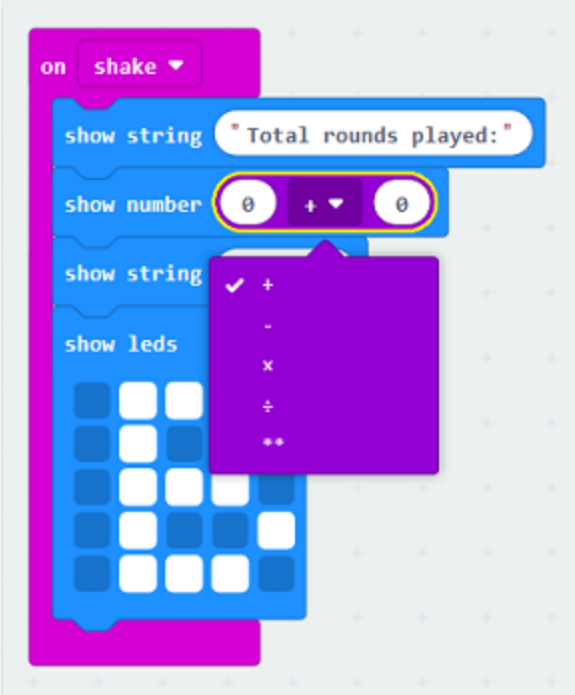
> First, display a string to show the player that the following sum represents the total number of rounds played.

Our program will add the values stored in the variables `PlayerAWins`, `PlayerBWins`, and `PlayersTie` and then display the sum of this mathematical operation.

The blocks for the mathematical operations adding, subtracting, multiplying, and dividing are listed in the Math section of the Toolbox.

Note: Even though there are 4 blocks shown for these 4 operations, you can access any of the four operations from any of the four blocks, and you can also access the exponent operation from these blocks.





Replace the default values of zero with the names of the variables we want to add together. Notice that because we are adding three variables together we need a second math block. First we add the values for `PlayerAWins` and `PlayerBWins`, then add `PlayersTie`.

Save, download, and try the program again to make sure that it runs correctly and displays the correct numbers for each variable.

Remember that the micro:bit is a device that processes input and displays it as output in some way. By storing values in variables, you can perform mathematical operations on that data that provides you with useful information.

What other math operations could provide valuable information from the values stored in these variables?

Examples:

Calculate and display a player’s wins and/or losses as a percentage of all rounds played.

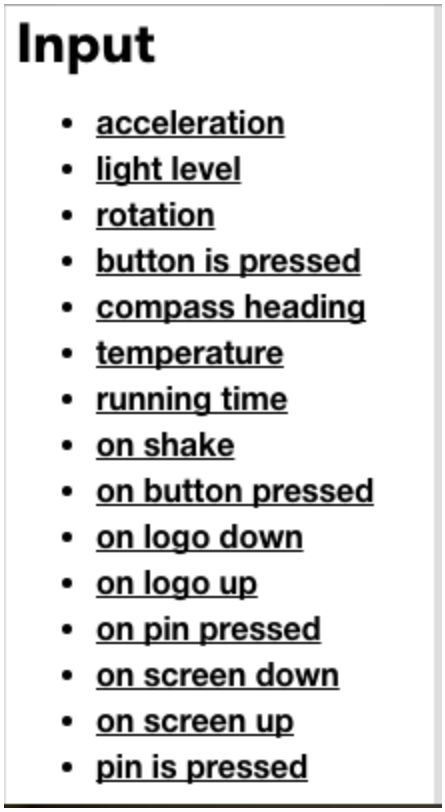
Calculate a display the number of tied games as a percentage of all rounds played.

4. Project: Everything counts

This is an assignment for students to come up with a micro:bit program that counts something. Their program should keep track of input by storing values in variables, and provide output in some visual and useful way. Students should also perform mathematical operations on the variables to give useful output.

Input

Remind the students of all the different inputs available to them through the micro:bit.



Project Ideas

Duct tape wallet

You can see the instructions for creating a durable, fashionable wallet or purse out of duct tape: Duct tape wallet. Create a place for the micro:bit to fit securely. Use Button A to add dollars to the wallet, and Button B to subtract dollars from the wallet.

Extra mod: Use other inputs to handle cents, and provide a way to display how much money is in the wallet in dollars and cents.

Umpire’s baseball counter (pitches and strikes)

In baseball during an at-bat, umpires must keep track of how many pitches have been thrown to each batter. Use Button A to record the number of balls (up to 4) and the number of strikes (up to 3).

Extra mod: Create a way to reset both variables to zero, create a way to see the number of balls and strikes on the screen at the same time.

Shake counter

Using the ‘On Shake’ block, you can detect when the micro:bit has been shaken and increment a variable accordingly. Try attaching the micro:bit to a lacrosse stick and keep track of how many times you have successfully thrown the ball up in the air and caught it.

Extra mod: Make the micro:bit create a sound of increasing pitch every time you successfully catch the ball.

Pedometer

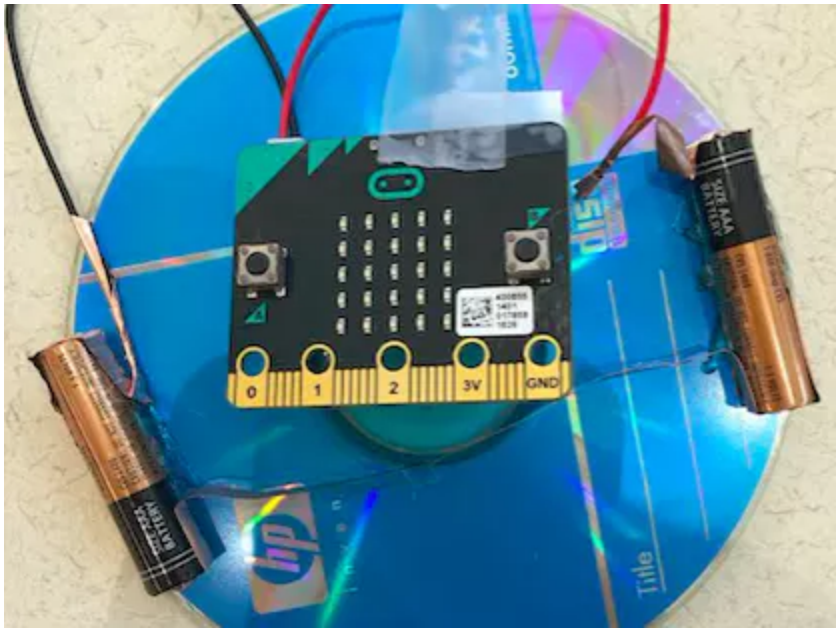
See if you can count your steps while running or doing other physical activities carrying the micro:bit. Where is it best mounted?

Extra Mod: Design a wearable band or holder that can carry the micro:bit securely so it doesn’t slip out during exercise.

Calculator

Create an adding machine. Use Button A to increment the first number, and Button B to increment the second number. Then, use Shake or Buttons A + B to add the two numbers and display their sum.

Extra mod: Find a way to select and perform other math operations.



Homemade top with micro:bit revolution counter

Duct tape wallet with micro:bit display

Baseball pitch counter

Process

In any design project, it's important to start by understanding the problem. You can begin this activity by interviewing people around you who might have encountered the problem you are trying to solve. For example, if you are designing a wallet, ask your friends how they store their money, credit cards, identification, etc. What are some challenges with their current system? What do they like about it? What else do they use their wallets for?

If you are designing something else, think about how you might find out more information about your problem through interviewing or observing people using current solutions.

Then start brainstorming. Sketch out a variety of different ideas. Remember that it's okay if the ideas seem far-out or impractical. Some of the best products come out of seemingly wild ideas that can ultimately be worked into the design of something useful. What kind of holder can you design to hold the micro:bit securely? How will it be used in the real world, as part of a physical design?

Use the simulator to do your programming, and test out a number of different ideas. What is the easiest way to keep track of data? If you are designing for the accelerometer, try to see what different values are generated through different actions (you can display the value the accelerometer is currently reading using the 'Show Number' block; clear the screen afterward so you can see the reading).

```
acceleration (mg)x
```

```
show ledsshownumber

forever
```

Reflection

Have students write a reflection of about 150–300 words, addressing the following points:

> What was the problem you were trying to solve with this project?
> What were the Variables that you used to keep track of information?
> What mathematical operations did you perform on your variables? What information did you provide?
> Describe what the physical component of your micro:bit project was (e.g., an armband, a wallet, a holder, etc.)
> How well did your prototype work? What were you happy with? What would you change?
> What was something that was surprising to you about the process of creating this project?
> Describe a difficult point in the process of designing this project, and explain how you resolved it.

Assessment

Competency scores: 4, 3, 2, 1

Variables

4 = At least 3 different variables are implemented in a meaningful way.

3 = At least 2 variables are implemented in a meaningful way.

2 = At least 1 variable is implemented in a meaningful way.

1 = No variables are implemented.

Variable names

4 = All variable names are unique and clearly describe what information values the variables hold.

3 = The majority of variable names are unique and clearly describe what information values the variables hold.

2 = A minority of variable names are unique and clearly describe what information values the variables hold.

1 = None of the variable names clearly describe what information values the variables hold.

Mathematical operations

4 = Uses a mathematical operation on at least two variables in a way that is integral to the program.

3 = Uses a mathematical operation on at least one variable in a way that is integral to the program.

2 = Uses a mathematical operation incorrectly or not in a way that is integral to the program.

1 = No mathematical operations are used.

micro:bit program

4 = micro:bit program:

* Uses variables in a way that is integral to the program
* Uses mathematical operations to add, subtract, multiply, and/or divide variables

* Compiles and runs as intended
* Meaningful comments in code

3 = micro:bit program lacks 1 of the required elements.

2 = micro:bit program lacks 2 of the required elements.

1 = micro:bit program lacks 3 or more of the required elements.

Collaboration reflection

4 = Reflection piece addresses all prompts.

3 = Reflection piece lacks 1 of the required elements.

2 = Reflection piece lacks 2 of the required elements.

1 = Reflection piece lacks 3 of the required elements.