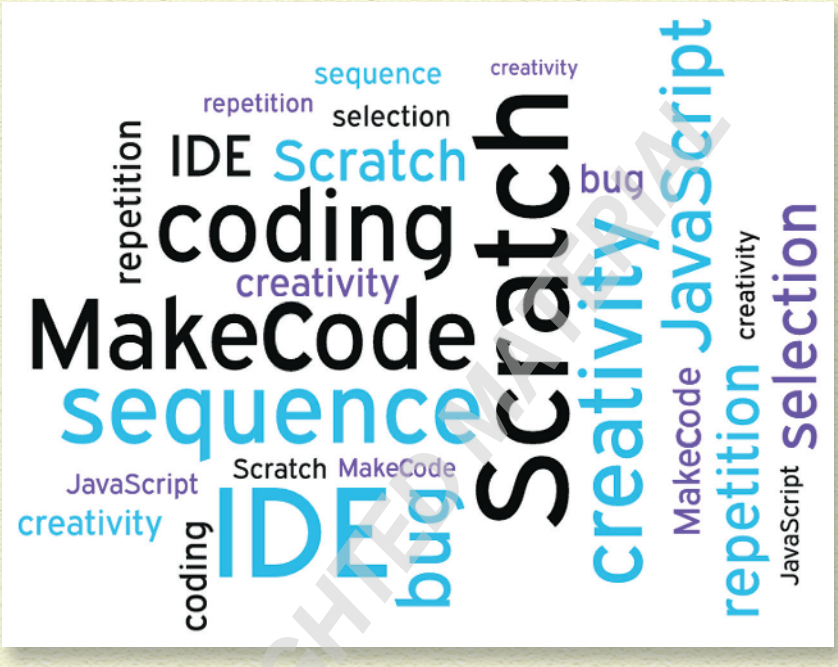
PROJECT 1 GET STARTED



IT USED TO BE ENOUGH TO READ, WRITE, AND DO MATH. These three skills were the tools we needed to communicate with other human beings. But now, the world is full of smart beings that aren't human — they're computers. Computers don't fully think for themselves *yet*, but they do "talk" with each other and with people through *computer programming languages.* Communicating with technology — speaking the language of computers — takes a new skill called coding (also called programming). This chapter gets you started with the basics you'll need to start coding fast!

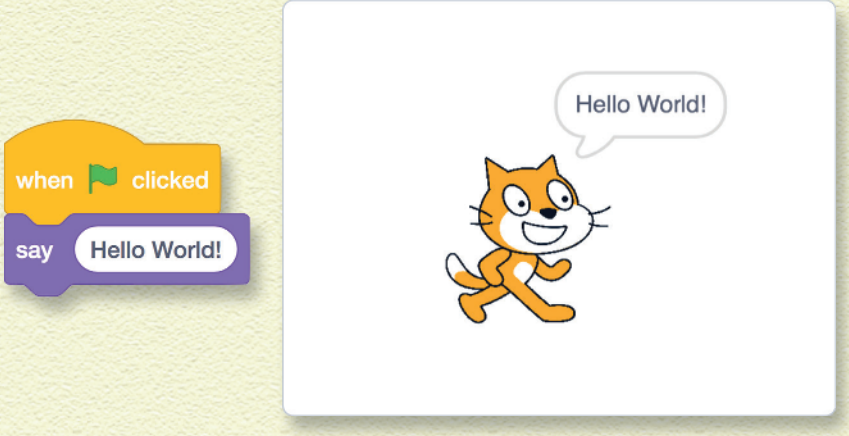
CODING QUICKSTART

Coding means writing instructions for a computer. The computer then uses the instructions to do a task.

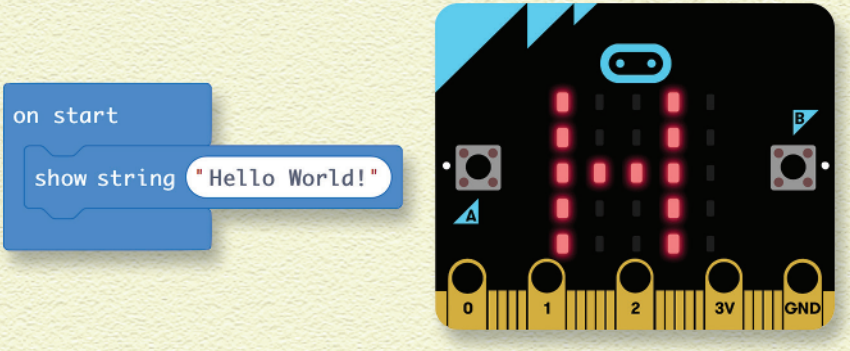
A computer programming language has *commands* (vocabulary) and *syntax* (grammar rules and punctuation) for communicating with a computer. As you write code, you put together the commands, using the correct syntax, in a logical way. The *logic* of your code is the order of the commands and the sequence of what happens due to various conditions. Put together, the instructions that you write and that the computer reads are called the *computer program*, or just the *code*.

A HELLO WORLD! EXAMPLE

The first program new coders often write prints the words *Hello World!* on the computer screen. Here's the code in Scratch and its *output* (what it shows onscreen).



Now here's the same code in JavaScript blocks built in MakeCode and the output on the micro:bit electronics board. Because the micro:bit can scroll only one letter at a time, the figure displays just the letter *H* at the beginning of the *Hello World!* message.

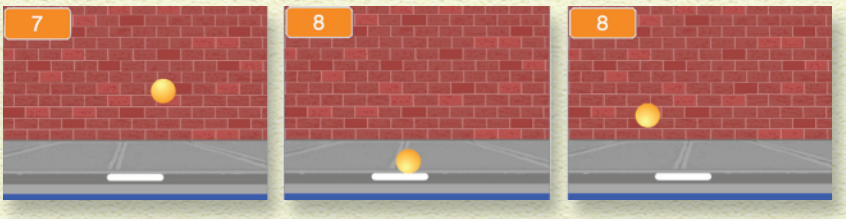


You'll be making little programs like this, and bigger programs too, in no time!

WHAT'S AN ALGORITHM?

A computer program has lots of parts, such as asking the user a question, doing something with that information, and then telling the user some response. Planning a computer program is a bit like telling a story or running a play in football. You have to put together and *execute* (run) the program in a certain order.

In each part of a program, you write small chunks of code to do different things. A chunk of code that does a task is called an *algorithm.* For example, in a paddle ball game, one algorithm you might use is bouncing the ball. If the ball and paddle touch, the player scores a point and the ball changes direction.



What algorithms do you see in games you play on your phone? One algorithm in a Yahtzee game is rolling the dice. An algorithm in a Space Invaders game is flying a spaceship across the sky. You also see algorithms in life, such as a pattern for vacuuming a room or a routine for driving to school. Look for algorithms in the apps you use and in your daily life.

SEQUENCE, SELECTION, AND REPETITION

The algorithms you write connect with each other to build your entire program. As coders, we have three fancy terms to describe how we build our algorithms and how they connect with each other: sequence, selection, and repetition. Here's what each means:

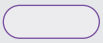
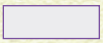
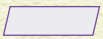
- **Sequence:** The order you run commands (or algorithms). Steps are executed one after another, in order. For example, when making pancakes, I run my algorithm for making batter before I run my algorithm for cooking the batter!
- **Selection:** Choose a path. For example, if you decide to go to a movie, you might then choose between an action movie and a comedy. That decision then directs you to new sequences: If you pick an action movie, you might then select among *Avengers, Pokémon,* and *Shazam!* You write selection code using *conditional* commands: if [this happens] then [do that]. Conditionals let you make as many paths as you need for your program.
- **Repetition:** Do something over and over. A *loop* tells the computer to run the same commands lots of times, without having to rewrite the commands. You already know how loops work: In a song, the drumbeat is looped to make the rhythm from the first note of the song to the last.

All computer programs have sequence, selection, and repetition. Check back here to refresh your memory of how each is used when coding a program.

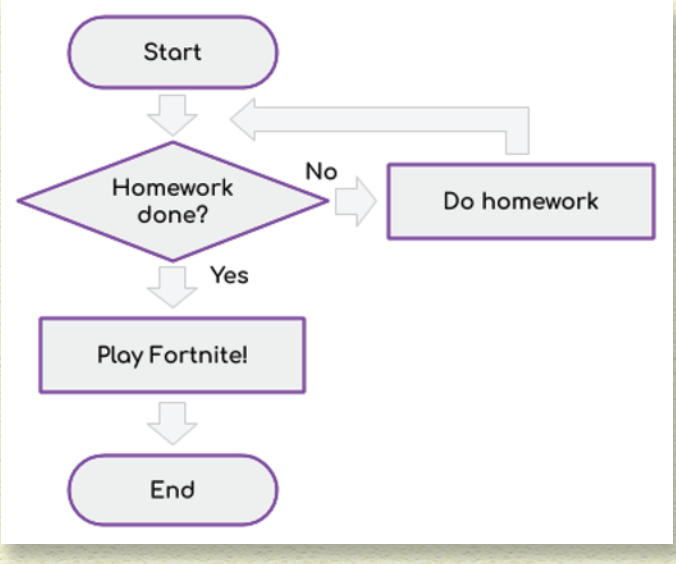
FLOWCHARTING

As you plan your programs, you can make a special drawing — called a flowchart — to show your plan. A *flowchart* is like a little map with special boxes and arrows that show the smaller parts of the whole plan. You can plan almost anything with a flowchart — a travel schedule, a recipe, a basketball play, or an app you're making. This table shows some flowchart symbols and what each symbol means.

SYMBOLS USED IN FLOWCHARTS

Symbol	Name	What It Means
	Arrow	Shows the sequence of the program
	Terminal	Starts or ends the program
	Process	Does a task, such as a math problem
	Decision	Selects yes or no to follow a new path
	Input/output	Takes in information or tells the user something

In this book, you plan each program with a simple flowchart. Here's a flowchart for a program to decide on an afterschool activity. Planning a computer program with a flowchart helps you think about the big picture first. You can get your thinking straight and leave the details of writing code for later!



PICKING A LANGUAGE

As a coder, you pick a coding language depending on what you want to build. Some languages are good for building programs you run in a web browser, others are good for building phone apps, and still others are best for writing code to control electronic circuits.

In this book, you use Scratch, a kids programming language, to make an animation, a toy puzzle, and a game for a computer or tablet. You also use JavaScript, a real programming language, to code electronic gadgets. In JavaScript, you make for the micro:bit board a thermometer and a game of card war. In both Scratch and JavaScript, you can work in *block mode*, snapping together code by using tile (puzzle) pieces to build your programs.

USING A DASHBOARD (IDE)

No matter what language you choose, you need a way to write and test your code in that language — a *coding dashboard.* Real coders use a special name for the dashboard: *integrated development environment,* or *IDE.* A coding IDE, or dashboard, is an all-in-one place where you do all the things needed to create your code and make it ready for end users. *End users* are the people who will use your programs.

The dashboards you will use are web-based, meaning you go to a website and do your work online. Scratch has its own dashboard, and JavaScript can be written in many different IDEs — in this book, you use the MakeCode dashboard to write code in block-based JavaScript. In the dashboard, you do these things:

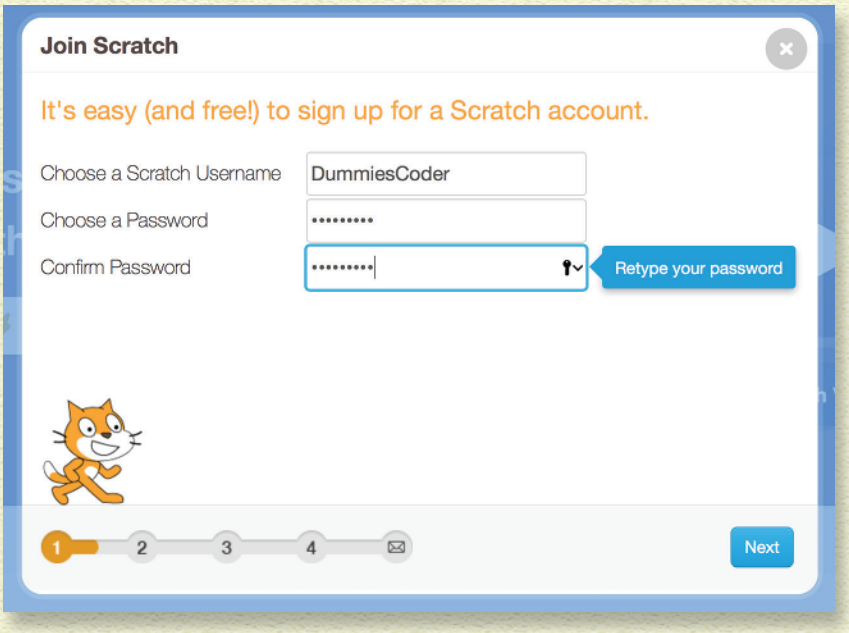
- **Write code.** You write code by dragging code tiles (individual blocks) to a workspace and snapping them together to build larger code blocks.
- **Create the screen layout.** Put together *assets* — such as sounds and graphics — to display onscreen in your program.
- **Compile your code.** *Compiling* means translating your code from a human-friendly form to one that the computer understands. Scratch and MakeCode do this for you automatically.
- **Debug.** You debug your code by editing it to fix any errors, or *bugs.* Bugs cause your program to not work.
- **Test.** You try out your code in a display window or simulator.

The following sections help you set up and understand the Scratch and MakeCode dashboards.

SETTING UP YOUR ACCOUNT IN SCRATCH

Scratch is free, but you need to set up an account before you can start coding. Just follow these steps:

1. **In any web browser, navigate to https://scratch.mit.edu.**
2. **On the Scratch home page, select Join Scratch.**
3. **In the Scratch dialog box, type a Scratch username and a password. Then click the Next button.**



4 **Type your birthdate, gender, and country. Then click Next.**

5 **Type the email of your parent or guardian. Then click Next.**

A screen appears letting you know that you are signed up to use Scratch and that a confirmation email has been sent to your parent or guardian.

6 **Your parent or guardian must open the email and confirm that you're permitted to share your work publicly on Scratch.**

If the adult doesn't confirm, you can still work in Scratch, but you won't be able to share your programs.

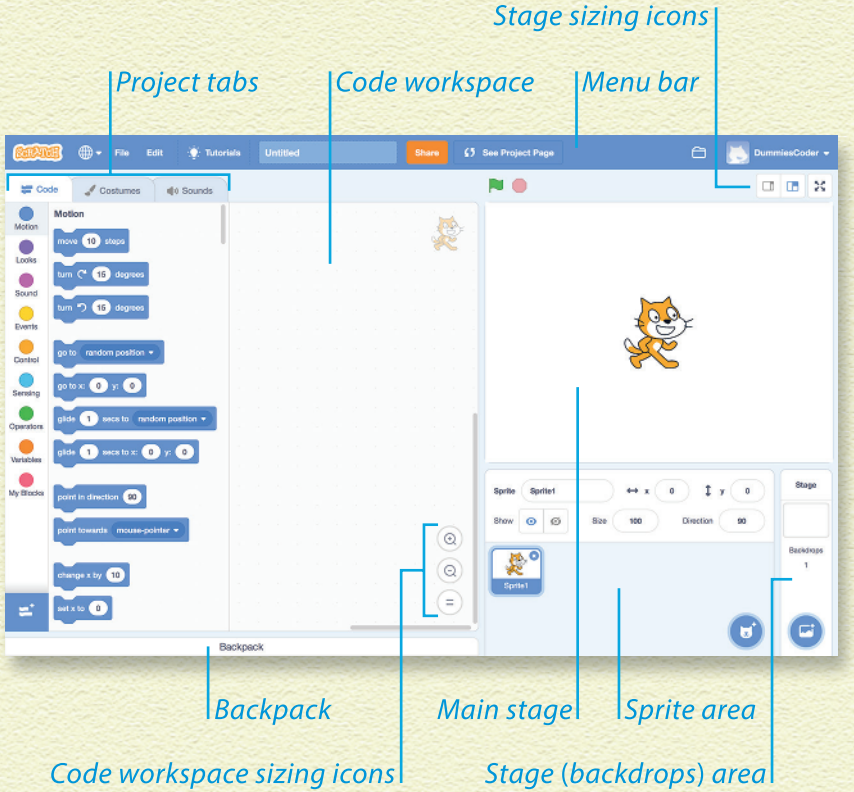
7 **Click OK on the final screen to complete the sign-up.**

After your Scratch account is set up, you can log in to your account at any time by clicking the Sign In button in the upper right of the Scratch home page and typing your username and password.

GETTING AROUND IN SCRATCH

After you set up your Scratch account, you are taken to the Scratch home page. Here are some of the things you'll see, and some of the actions you can perform.

Select Create in the menu bar on the Scratch home page to open the Scratch dashboard. In the Scratch dashboard, you see a new, blank project. This is the same screen you see if you choose File➪New when working in Scratch.



The *menu bar* at the top of the Scratch dashboard features just a few choices. These are the most important:

- **File ➪ New:** Create a new project.
- **Edit:** Turn Turbo mode on or off. When on, this mode runs the program superfast.
- **Project name field:** Name your project.
- **Share button:** Share your project.
- **See Project Page button:** Go to the project page for your current project, where you can tell users how to

use your program. On the project page, click the See Inside button to get back to the Scratch dashboard.

- **My Stuff folder icon:** Open projects you've created.

The *tabs* on the left side of the Scratch dashboard change depending on whether you're creating code for a *sprite* (an object in Scratch) or for the *main stage* (the backdrop where your sprites live):

- **Code tab:** Command categories and their blocks
- **Costumes tab:** Sprite costumes and the costume editor
- **Backdrops tab:** Backdrops for the stage and the backdrop editor
- **Sounds tab:** Sounds and the sound editor

The *workspace* is the place where you drag command blocks and then assemble those command blocks together to make larger *code blocks.* Click the *sizing icons* to zoom in or zoom out as you build code.

The *backpack* area at the bottom of the Scratch dashboard is a place where you can save, store, and reuse project items, sharing them among projects. Just drag a code block, sprite, costume, or sound into the backpack and it will be available in all projects! Drag an item from the backpack and into a project's workspace to use it. Delete an item in the backpack by Ctrl-clicking (Mac) or right-clicking (Windows) the item.

The *main stage* is the large window that shows you what your user sees. Here, you can see the sprites and the backdrop. Click a *stage sizing icon* to resize your view of the stage. Above the stage are the *green flag* and *stop* icons; use these to start or stop most projects you create.

The *sprite area* shows the sprites (objects) in your project. Each time you start a new project, the Scratch Cat sprite is added by default (something the computer does automatically) and named Sprite1. You can find out more information about Sprite1 in the Sprite area, just below the main stage. The Sprite area tells you the sprite's position on the main stage (the *x* and *y* values), whether the sprite is showing or hiding (the eyeball icons), its size, and its direction. To delete a sprite, click the X in the corner of the sprite icon. To add a new sprite, click the *choose a sprite* icon (which looks like a kitty head with a plus sign).

The small *stage,* or *backdrops* area, is located in the lower-right corner of the Scratch dashboard. Here, an icon shows the current backdrop. Clicking this icon makes the stage active, which means you can write code for the stage, change which backdrop shows in the background, and add sounds to the background.

WARNING

To write code for a sprite or the stage, you first click its icon to select it, or make it active. The active icon has a blue outline. Only one sprite or the stage can be active at a time, so pay attention to which of these is currently selected.

USING THE CODE TAB IN SCRATCH

The Scratch dashboard in the previous figure is shown with the Code tab selected. This tab has all the commands you can use in your programs. You use this tab for both sprites and the stage. It is organized by command categories, as follows:

- **Motion (blue):** Commands to tell sprites (objects) how and where to move. (The stage does not have motion commands.)

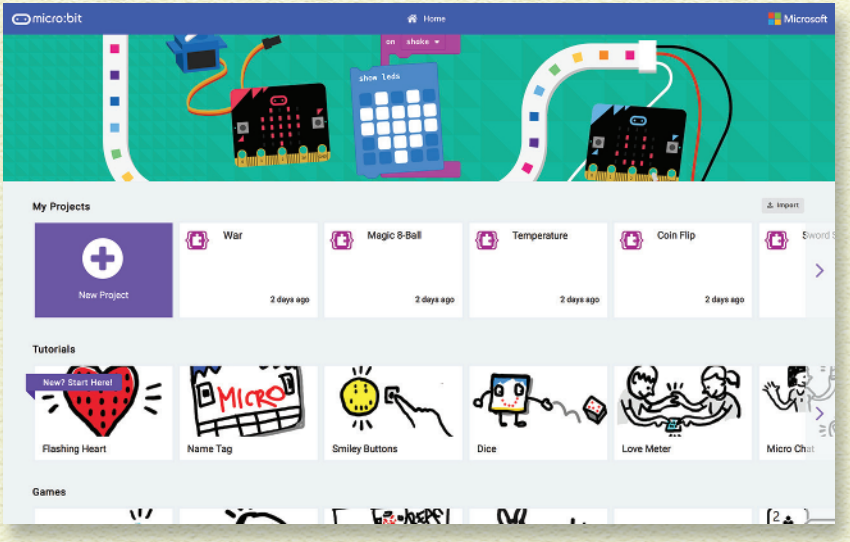
- **Looks (purple):** Commands to change the costumes of sprites and change which backdrop is on the stage background.
- **Sound (pink):** Commands to play music and sound effects.
- **Events (yellow):** Commands to start and end code execution.
- **Control (light orange):** Commands to select code or repeat code for execution. (See the "Sequence, selection, and repetition" section for details.)
- **Sensing (light blue):** Commands to sense color, sound, position, and user input.
- **Operators (green):** Commands for math and logic operations.
- **Variables (dark orange):** Commands to create and change variables.
- **My Blocks (red):** New commands you define for your project.
- **Add extension (wavy lines):** Access to additional commands you can add in Scratch 3.0, such as text-to-speech and language translation.

To write code for a sprite or the stage, click a command category. Then drag one block (command) from the category and drop it into the workspace. Add the next command by dragging a new block below the previous one and then snapping it (bringing it close) to build a code block. Remove a command by clicking it and dragging it back to the command categories (anywhere). To *execute*, or run, the code block, click the block. The code block will light up with a bright yellow outline, and the commands in that block will execute.

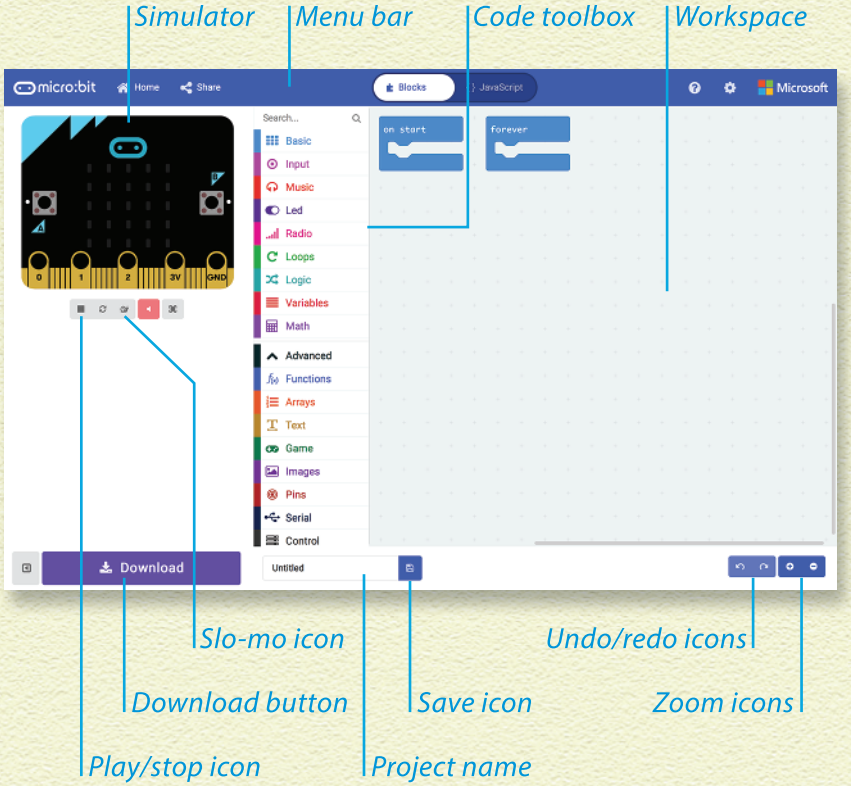
Scratch automatically saves your projects in the cloud, but you can also save at any time by choosing File➪Save Now from the menu bar.

GETTING AROUND IN MAKECODE

MakeCode doesn't require you to create an account. Access the MakeCode site shown here at https://makecode.microbit.org. This is your page for creating a new project, opening projects you've made, and seeing sample projects and tutorials for lots of other projects you can program for the micro:bit. You can get back to this page at any time by clicking the Home button at the top of the page.



When you create a new project in MakeCode, you see the MakeCode dashboard (IDE) shown here.



The *menu bar* at the top of the MakeCode dashboard features just a few choices. These are the most important:

- **Home button:** Return to the MakeCode home page.
- **Share button:** Share your project.
- **Blocks/JavaScript toggle:** Choose to code in blocks or JavaScript text mode.

The micro:bit *simulator*, which is the window on the left, shows what your user sees. It acts the way a micro:bit does, except it's digital (not a real, physical electronics board). Here, you can see and run your program to get

everything working before you transfer your program to the micro:bit. The simulator icons you can click are

- **Play:** Run your program. (This icon looks like an arrowhead.)
- **Stop:** Stop program execution.
- **Restart:** Reset the program to the start.
- **Slo-mo:** Slow the rate of program execution, a helpful tool when debugging. (This icon looks like a snail.)
- **Audio:** Mute or unmute sound.
- **Full screen:** Make your micro:bit simulator appear in a large format on its own screen.

The *workspace* is the place where you drag command blocks and put commands together to build your *program.* Click the *zoom in* or *zoom out* icon in the bottom-right corner to adjust your view. If needed, you can click the *undo* or *redo* icon in the bottom-right corner to edit your code.

The *debug console* is a window that appears below the simulator if you have an error in your code. This console gives you clues about mistakes in your program and where to look to fix them.


TIP

In this book, you write JavaScript code in block mode using command blocks (also known as tiles). But you can also write in text-based mode by clicking the JavaScript button in the menu bar of the MakeCode dashboard. To switch back to blocks mode, click the Blocks button.

USING THE CODE TOOLBOX IN MAKECODE

MakeCode's *code toolbox* provides coding commands for you to build your micro:bit programs. Here are the categories of commands:

- **Basic:** Commands to show images or text by using the LEDs, as well as the on start and forever commands.
- **Input:** Commands to read user input, such as a button click, a shake of the micro:bit board, or an electrical signal from one of the pins.
- **Music:** Commands for playing music. Sound can play in the simulator, but the micro:bit does not have a built-in speaker, so you need to hook up headphones or a speaker to make the micro:bit produce sound.
- **Led:** Commands for making the LEDs light up individually.
- **Radio:** Commands for having micro:bits talk to each other by using a radio frequency.
- **Loops:** Commands for doing repeats.
- **Logic:** Commands for making decisions (conditionals, comparisons, and Booleans).
- **Variables:** Commands to make and change the value of a variable.
- **Math:** Commands for doing math.

To write code, select a command category and then drag and drop one block (command tile) into the workspace. Add the next command by dragging it below the previous one and then bringing it close to snap the tiles together, building your program. Delete

a command by clicking it and dragging it back to the code toolbox. (Or in text-mode, select the command you want to get rid of and then click the Delete key on your keyboard.) To *execute*, or run, the program on the simulator, click the *play* icon.

To *save* your program, type the name of your program in the Untitled field at the bottom of the MakeCode dashboard and then click the *save* icon.

MakeCode for micro:bit offers Advanced commands, with lots of other command categories. These commands let you build even more complex programs. If you want to challenge yourself, check them out!

FIXING ERRORS

Your code probably won't be perfect the first time you write it. Whether you're just starting out coding or you've been programming for a while, you — like everyone else — will make mistakes when writing code. As a coder, you will spend a lot of time fixing those mistakes, or *bugs*, in your programs. (They're called *bugs* because one of the "mistakes" in the early days of computing was an actual bug in the computer's electronic circuitry!)

You will need to find your errors and fix them before your programs can run. Fixing mistakes in your code is called *debugging*. Here are the two types of bugs you will need to remove:

- **Syntax errors:** When the form of the code is wrong — you spelled something wrong or messed up your punctuation — you get a *syntax error*. Block coding makes it hard to make syntax errors because there's almost no typing and the blocks fit together in specific

ways. But if a syntax error does happen, you must fix this mistake or else the program won't run.

» **Logic errors:** When the behavior of the code is wrong, you get a *logic error*. For example, if the game score goes down each time a hero defeats an enemy, you may have coded a score variable to decrease, not increase. The code runs because the syntax is correct, but the program doesn't do what you want it to do. At that point, you need to figure out what is going wrong, and then find the part of your code that controls the behavior that is not working correctly.

Testing your program means running it to see if it works the way you want it to. The following sections give you some strategies for debugging your code in Scratch and in MakeCode.

DEBUGGING IN SCRATCH

Scratch helps stop you from making syntax errors because you don't type the commands. Instead, you drag command blocks into the code workspace. You don't have to worry about misspelling a command or forgetting to add a semicolon because Scratch prevents you from making these types of errors. Instead of typing a premade element such as a variable name, for example, you select it from a drop-down list.

When coding in Scratch, you're likely to make mistakes in the logic of your program. Because you see these mistakes when you run your program, they are called *runtime errors.* To fix this type of error, you try to find the exact place where something is going wrong in the execution of your program. Then you change the code to correct your error (or errors).

Here are some ways to locate and fix runtime errors in Scratch:

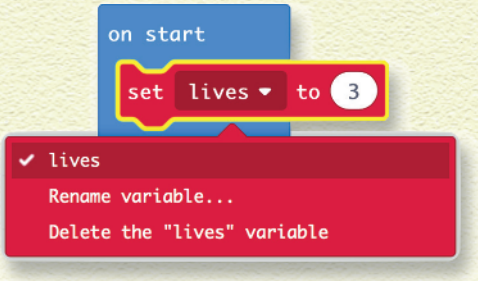
- **Focus on one sprite at a time.** If a sprite is working the way you want, move on to the next sprite and check its operation. (In many projects, you also write code for the stage, and you can use this method to check the stage as well.)
- **When you find a sprite with a runtime error, run the program a few times to see what the sprite is doing.** Try to figure out exactly what is going wrong and when that behavior occurs.
- **Look carefully at the code for the sprite with the runtime error.** Step through the code one command at a time. Separate some of the code (remove it temporarily by dragging it to the side, away from the event command). Then add sections of your code back in, one command at a time, and test the operation of the sprite after each addition. You should be able to identify where the error is occurring. At this point, fixing the error is usually easy.

Here's an example of a logic, or runtime, error in Scratch. I want my butterfly to fly toward the left of the screen, but it's flying toward the right. I think to myself, "Hmmm, my butterfly is flying, and he's flying at

the speed I want. But he's going the wrong way. Maybe I set my direction wrong."

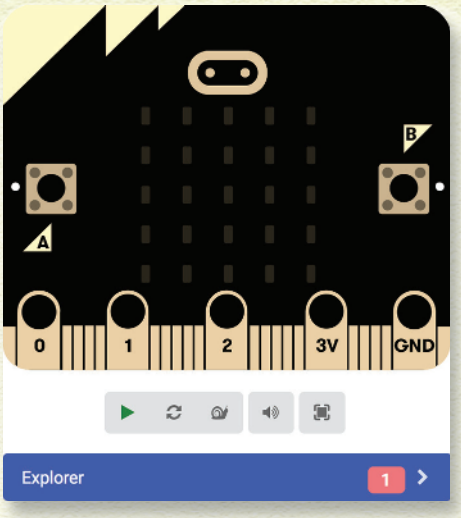
So I look at the part of my code that has to do with direction. I see that I have set the direction to positive 90, pointing the butterfly to the right. I should have set the direction to negative 90 (by typing `-90`), pointing the butterfly to the left. So I type the correct value and test my code to make sure it is fixed.

DEBUGGING IN MAKECODE



The block mode of MakeCode is like Scratch — it helps keep you from making syntax errors. Because you use premade blocks (also called command tiles), you can't type misspellings or forget to include punctuation.

When coding in MakeCode using blocks, any errors will probably happen at runtime. If you run a MakeCode program in the micro:bit simulator and get an error, an Explorer window appears just below the micro:bit simulator. The MakeCode Explorer window displays the number of errors in the program.



You can click the Explorer window to expand it and view more information about the execution of your program and the errors. But be aware that it's a bit hard for new coders to read and understand the Explorer window!

Your best bet for fixing runtime errors in MakeCode is to run the program a few times and trace the code as it runs, step-by-step. Try to pinpoint the exact moment when an error takes place, and then go to that part of the code and look closely at it. Change the code to fix your error (or errors).


TIP

Click the slo-mo icon in the simulator to slow the program speed, making it easier to trace the action.

Remember, half of coding is writing the code — and the other half is debugging it!

GETTING HELP

This book covers just a few key commands in Scratch and MakeCode. For more help, the Scratch home page has a Search field in the menu bar, where you can type a search term to get information on that topic. In MakeCode, go to https://makecode.microbit.org/reference to look up how to use a command. Also, when working in the MakeCode dashboard, hover your cursor over any command tile to see a tip with information on the command.

Additionally, both Scratch and MakeCode offer tutorials and project samples at https://scratch.mit.edu/ideas and https://microbit.org/ideas. Every example that you view, play with, and code yourself is a learning opportunity!