Funbytes

Book Design by HMDPUBLISHING

Contents

For my sister, Enda

Introduction: Welcome to The World of Coding!

Why Should I Care?

It's a weekday, so your phone's alarm woke you up. You get mad at your alarm and you want to break it, but you are wise so you don't. You walk to the bathroom, wishing it was a weekend. You brush your teeth (hopefully) and you go back to your room to change into your school outfit. After you change, you get into your mom's car, and your mom starts playing your favorite song from the car's music player.

You arrive at the school and your mom drops you off. At school, you catch up with your friends on what you did over the weekend and share some pictures of fun stuff that happened. The bell rings, and you all head to class. Since notebooks aren't cool anymore you open up your Chromebook to take some notes, and the teacher uses the projector to teach you. Then, after 5 boring periods, the final bell rings, and school is over.

You and your friends decide to go out for a meal, and you agree on McDonalds. When you arrive, you decide to order a BigMac (because you are really hungry), and the cashier uses a special device to handle your order. You all sit down with your friends and feast on one of the McDonalds tables.

You go home after this busy (but fun) day, and after doing some homework, you boot up your PC and start playing Fortnite to relax. You play for 2 hours and decide you are too tired to do anything else and you just go to sleep.

Now, why did I walk you through an imaginary day? It's because I wanted to show you the importance of technology in our lives. In the 21st century, technology is in every part of our lives: our phones, car software that allows you to play music, Chromebooks, projectors, McDonald's cashier devices… It is everywhere. No matter where you look, there is technology.

You might now be wondering, who is behind all this? The answer is programmers. Programmers and coders build the software that allows you to do everything you just did that day. Someone created that Chromebook to allow you to take notes or someone coded Fortnite for you to play. A line of code is behind every interaction you have with technology, no matter how big or small it might be. The reason why you should care is because of this. It is because coding allows you to create these important interactions for everyone to enjoy. You can create whatever you want. A quiz app about Fruit-Ninja? A website about how much you like cute pandas? The next Google? Whatever it may be, I can promise you that it can be done through coding. The book that you are holding is the first step in your coding adventure. So, let's dive right into this fun world!

CHAPTER 1:

Getting Started

Coding vs Programming

Now, you might hear coding and programming used in place of each other. While this is usually not a problem, and I use them in place of each other in this book, there is still a small difference.

Imagine you have a friend, Joe, who is a car expert. He wants to make a car. Exciting, right? Cars are made up of different parts, such as a steering wheel, doors, and an engine. This is no problem, since he is an expert at making those things, so it just takes Joe a day to make them. Then comes the hard part. Joe has to assemble all of these things in a way that makes the car work. For example, he can't just put the steering wheel in the backseat, or the engine in the middle of the car. While that would be really cool, it isn't really functional. So, he carefully plans out what parts go where, and he executes that plan. He works for weeks, and finally, he gifts you the car he made.

In this little scenario, Joe making the car parts is “coding”. Joe assembling the different car parts that he made is “programming”. Coding is the act of writing code. So for example, this gentleman here is coding.

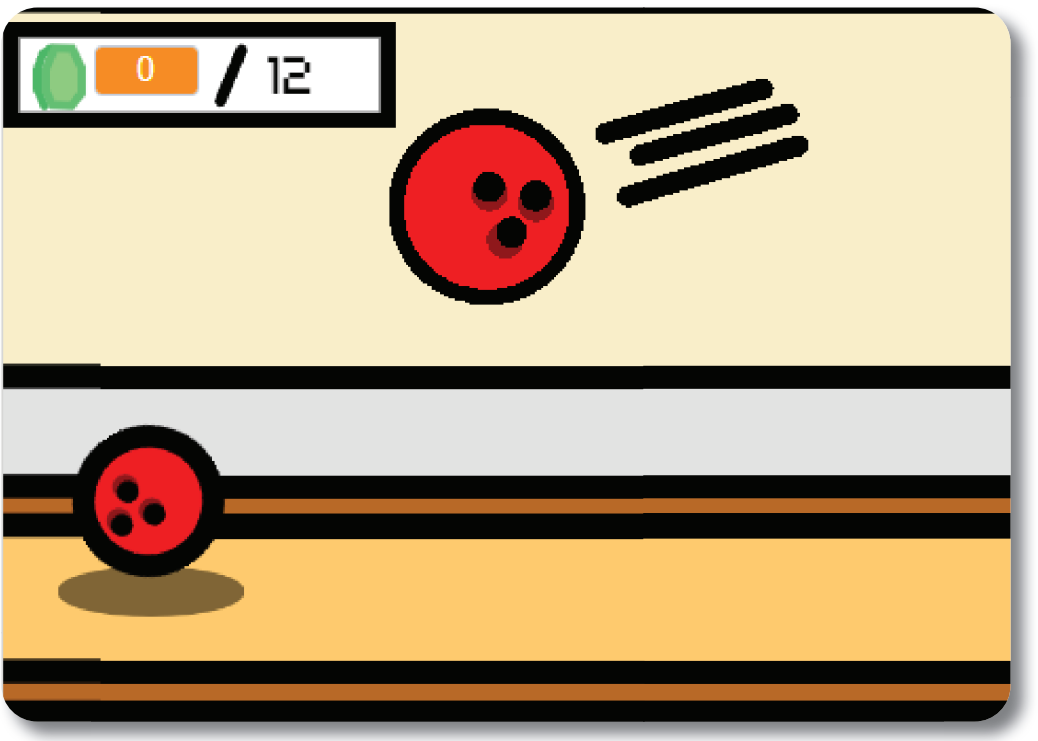
Programming, however, is the act of organizing and making that code actually do something.

Meet Your Coding Friend: Scratch!

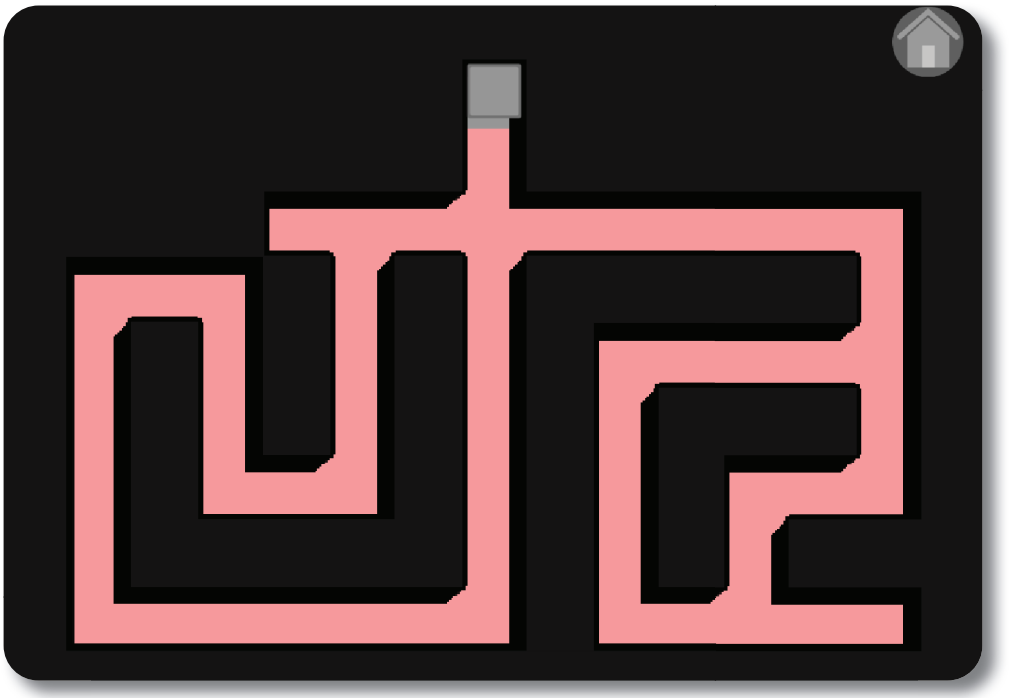
In your coding journey, you will encounter countless different coding languages. These are just like real languages, but all of them are used to talk to a computer instead of a person. The one we will be using in the first part of this book is called Scratch, and here is what they look like:

They are really excited to meet you too. You can use Scratch to create anything you want, but in this book, I will just use it to teach you the basic coding stuff that is used in every coding language. Here are a few examples of what you can make with Scratch.

Unusual Bowling by UjaanRoy

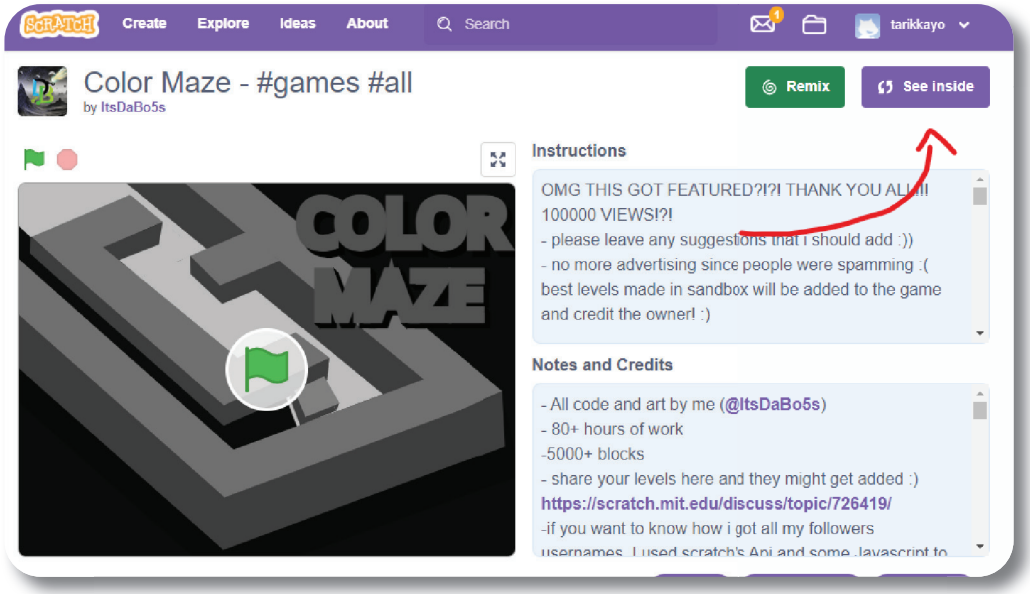


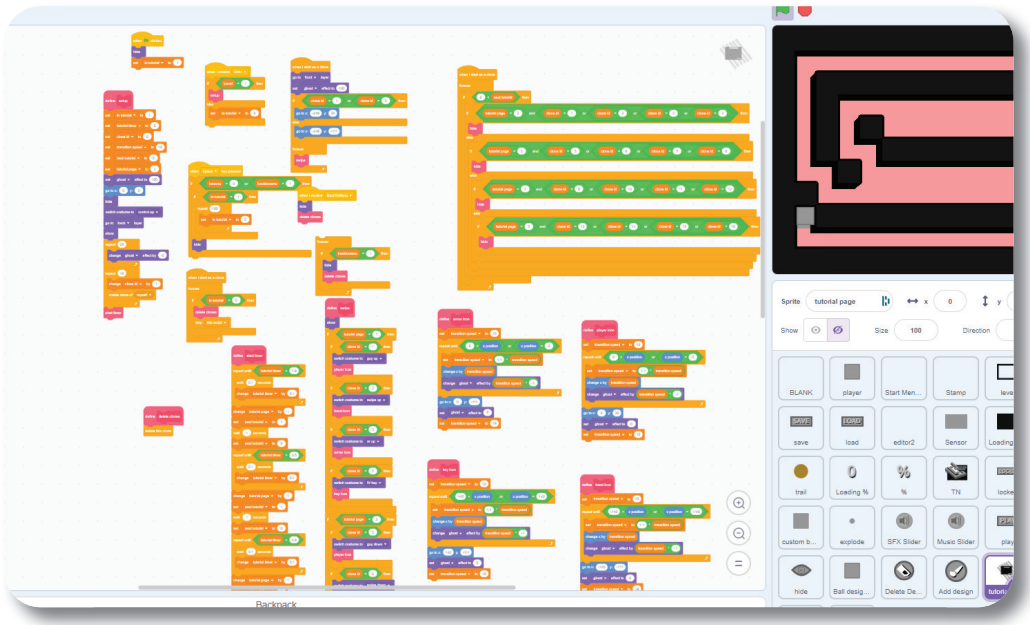
Color Maze by ItsDaBo5s



Marshmallow Madness by cheekybubba5

You can find more games at https://scratch.mit.edu/, and even take a look at how they are made using the “See Inside” button on any of the project pages.





This is how a project looks on the insides. Don’t let this intimidate you, since you will have a strong understanding of how almost

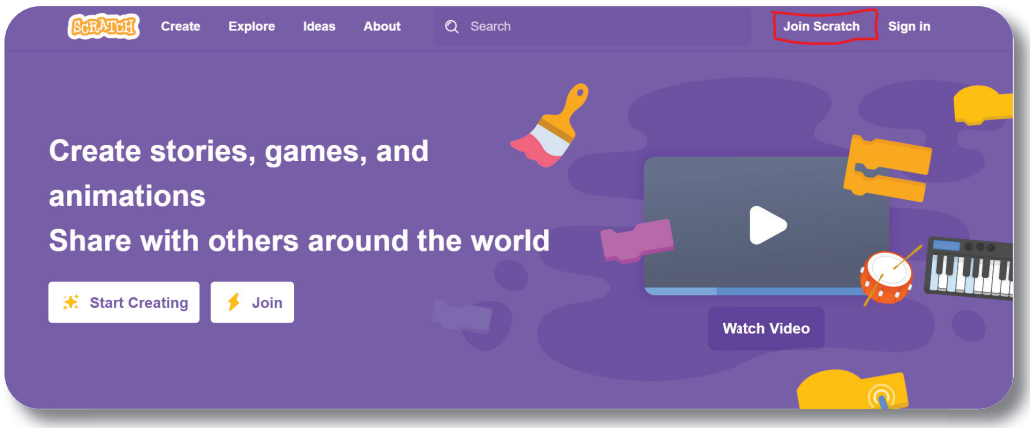
all of those blocks work at the end of the Scratch section of the book.

Setting Up Scratch

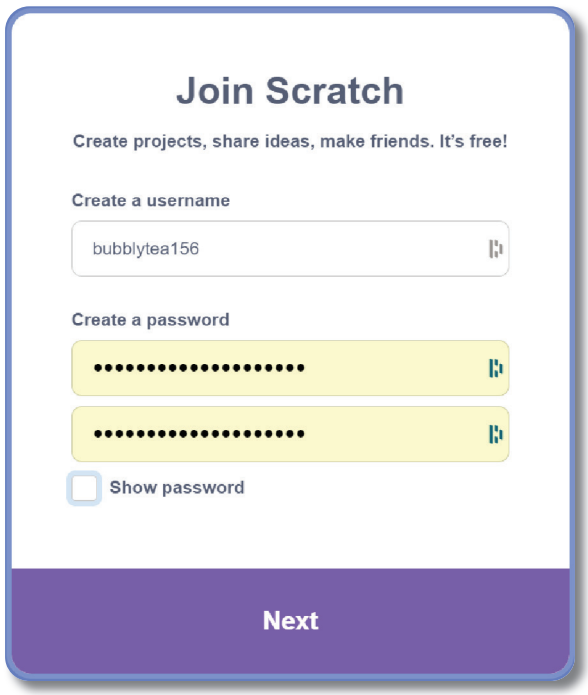
In this section, we will set up Scratch.

I need you to go to the main page of Scratch at https://scratch.mit.edu/ (with a parent's permission). Here, you will create your account for Scratch.

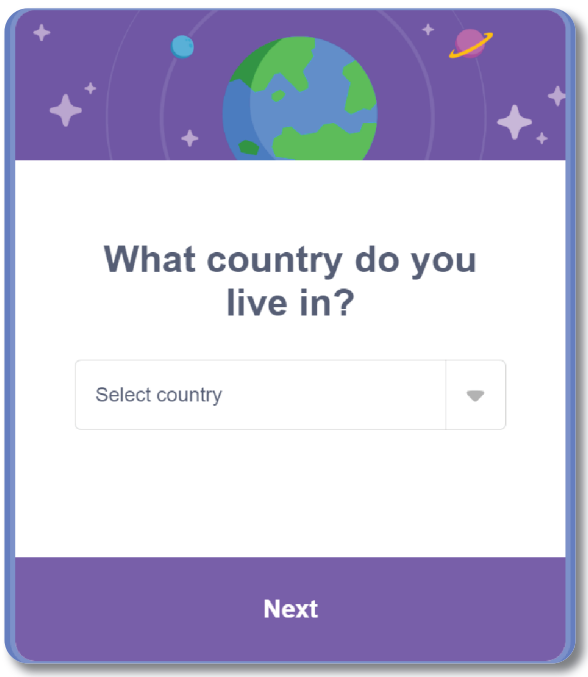
If you already have an account, click on "Sign In", and you can skip the next few pages that explain the account creation process. If you don't have an account yet, click on "Join Scratch".



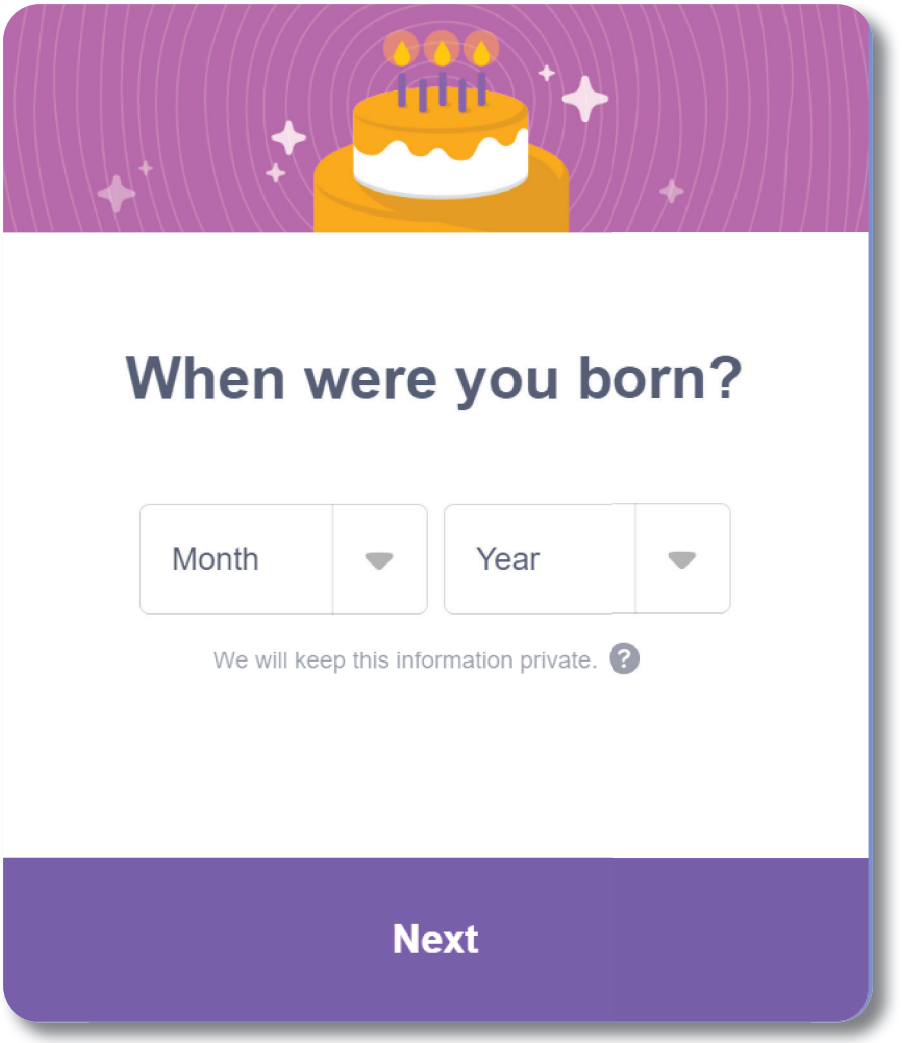
Now, pick a username and a password. Keep in mind that your username shouldn't be your real name.



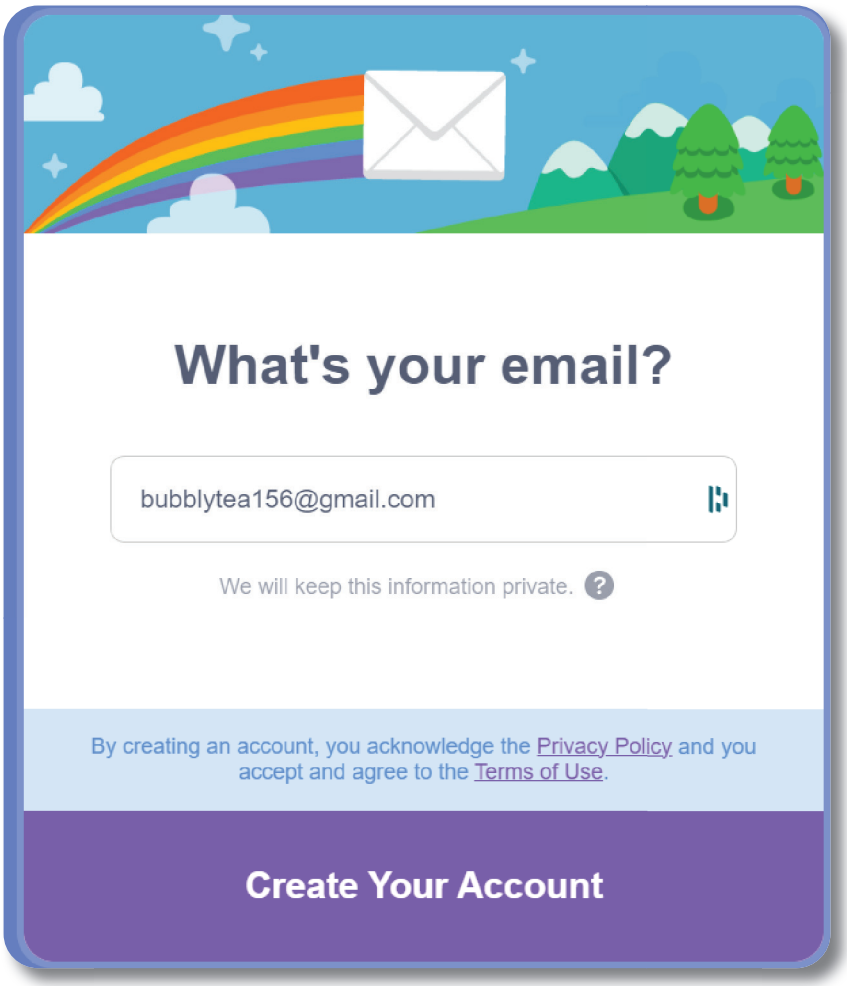
Now, select your country.



For this next step, you need to enter your birth date. Don't worry, this will not be visible on your profile. Make sure that this is accurate and not like the 10th of May 1895.



Now, enter in your email



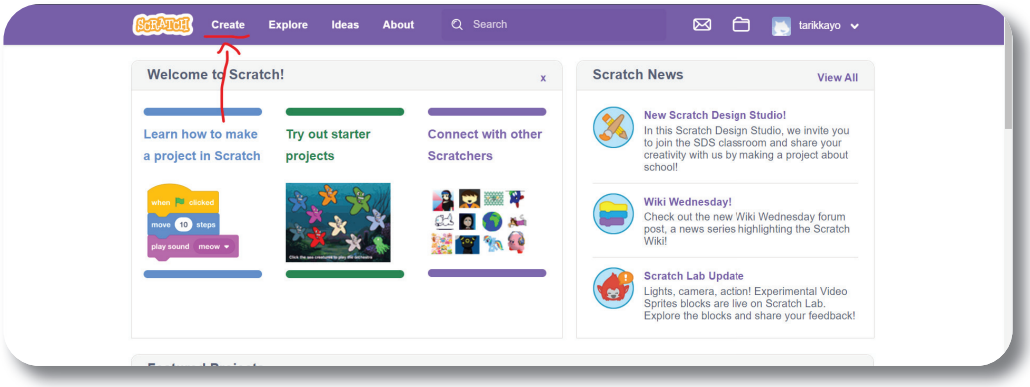
You need to verify your email to share your projects, so go ahead and verify your email.

You should be all set now! Let's start creating with Scratch!

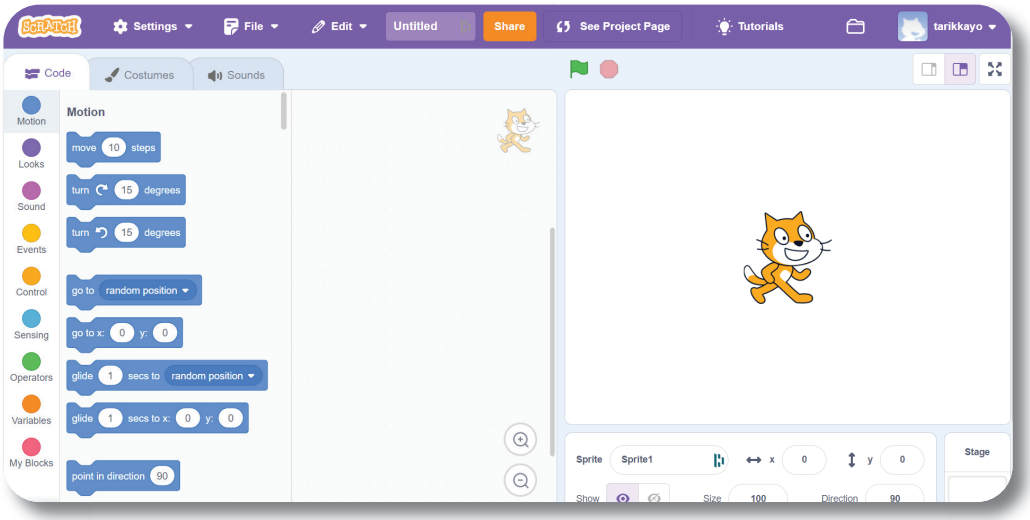
Hello World!

Now that you have your Scratch account, we are ready to start creating.

Go ahead and click on "Create" on your navigation bar.

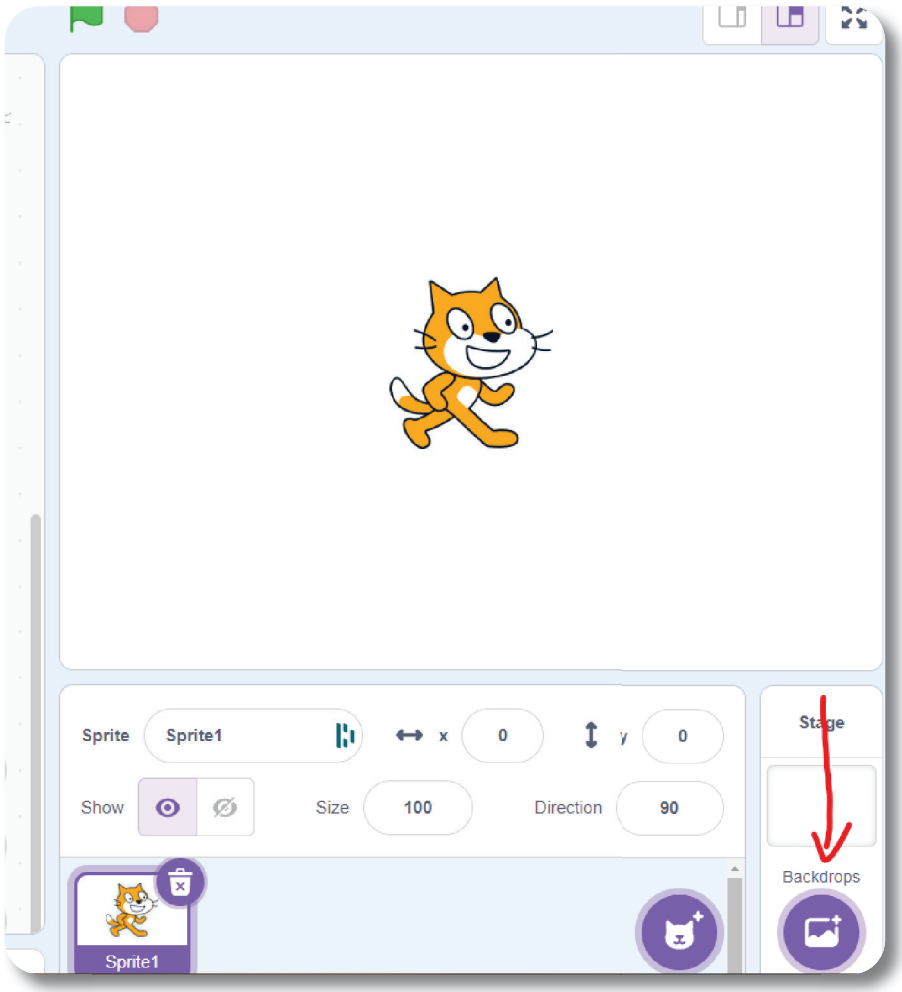


Once you do, the website will send you over to the “Scratch Editor”. This is the place where we will be creating our games and learning the basics of coding. Once the editor is loaded, you should see something like this:

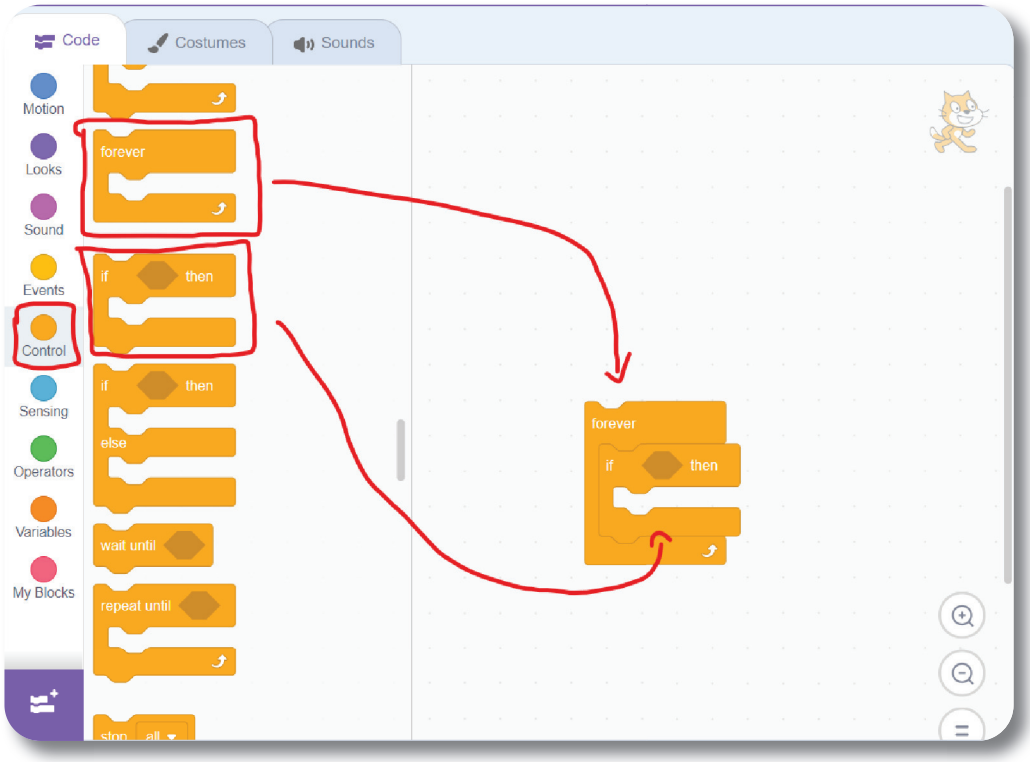


The blocks on the left side are called “blocks”. Creative, right? Anyways, as you can see, there are different categories of blocks, 8 to be exact. We won’t worry about the last category called “My Blocks”.

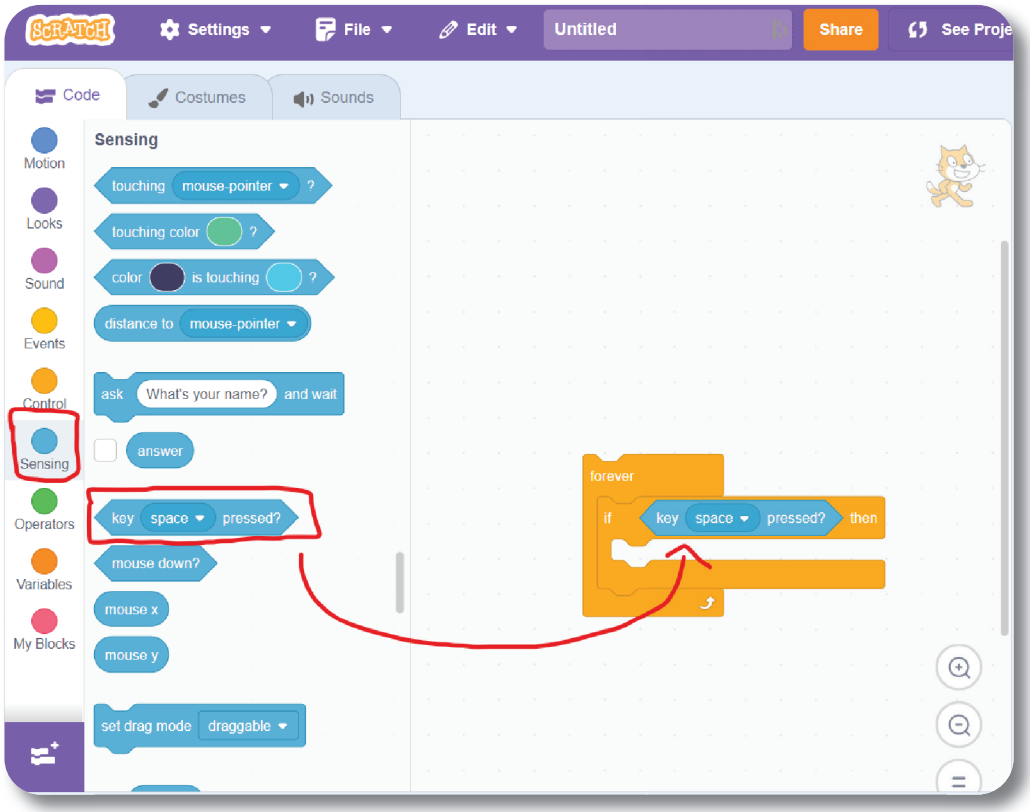
Now, let's do some coding! First, go ahead and pick a nice backdrop. I like the Arctic one.



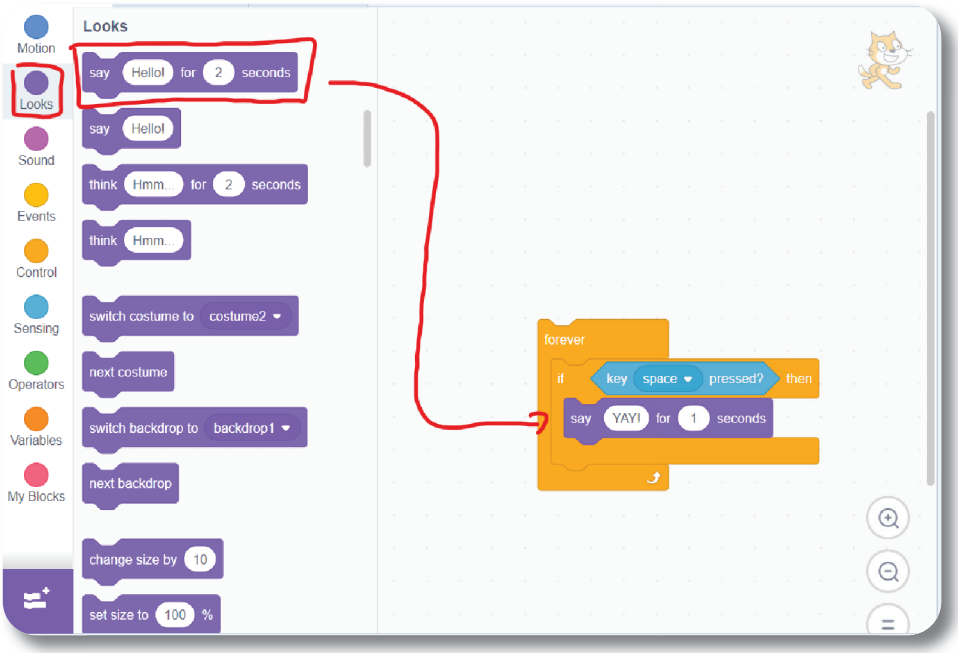
Next, I want you to go to the “Control” section, and drag and drop a “forever” block onto the screen, Then, drop a “if” block inside the forever block. Don’t worry about what any of this means for now, I will explain them in detail in later chapters.



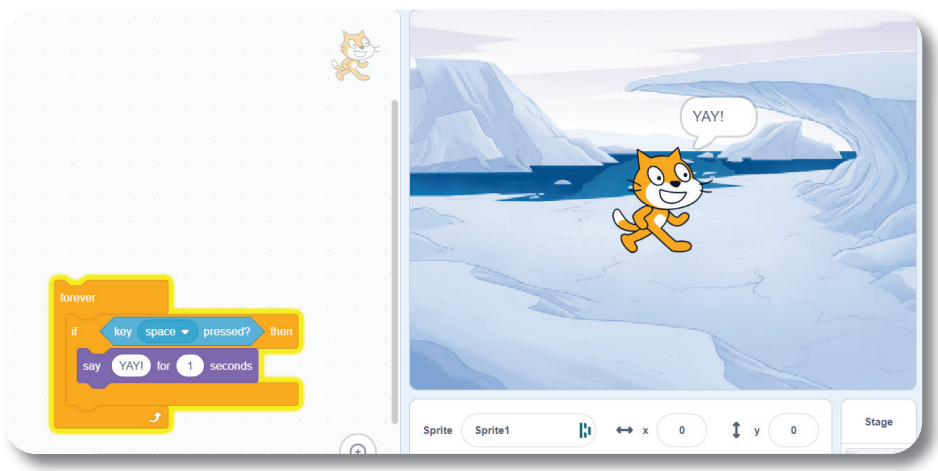
After that, go over to the “Sensing” section, and drag and drop a “key ___ is pressed?” block. Again, I will explain this in detail in later chapters.



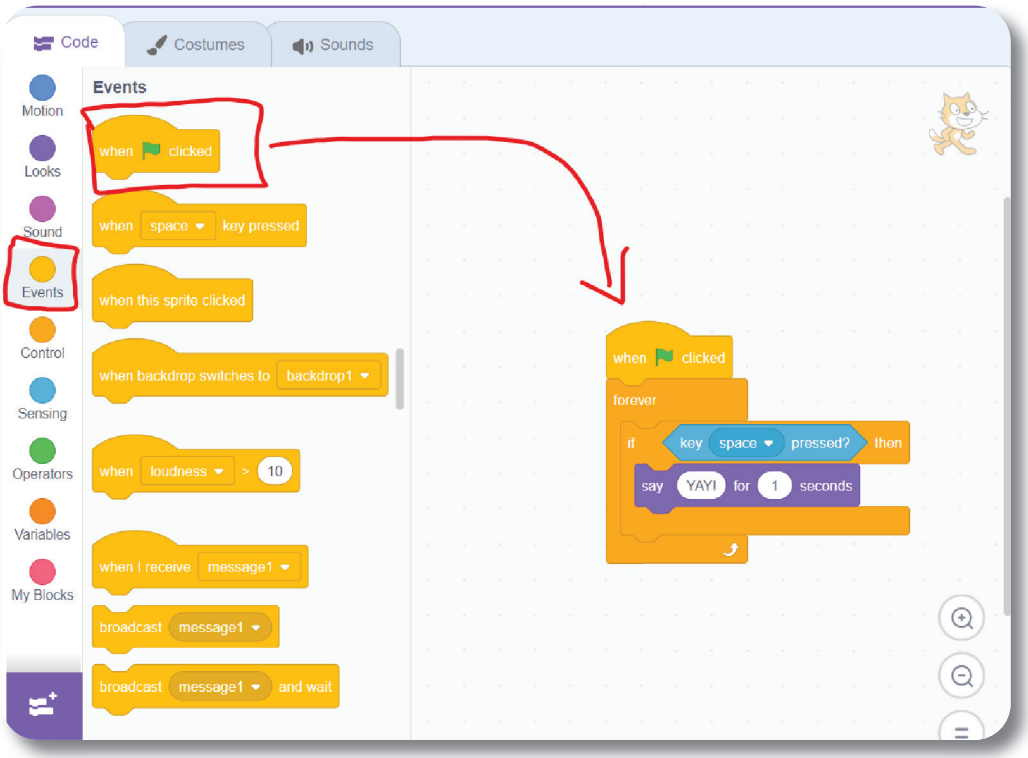
Now, go over to the “Looks” section and put a “say ___ for ___ seconds” block inside of the if block. You can switch out the “Hello” and “2” for whichever values you want. I will set my text to “YAY!” and the duration to “1”.



We are almost there. You can try this out right now by clicking on the “forever” block, where the block will have a yellow glow. That means that the code is working, and if you press “space” (or whatever you set it to be), your character should say whatever you want it to say.



You can click on blocks like this individually to test them out, but for now, we don't need to. Another thing you can do is to add the  block on top of the forever block to make the process simpler. You can find it in the "Events" section.



And now, when you click the green flag button, your program will start. When your program is running, if you press the key you said (space in this case), your character will say "YAY!". Exciting, right?

Now that you have done this, let's start exploring what algorithms are!

CHAPTER 2:

Understanding Algorithms

What is an Algorithm?

I know you are wondering: What is an algorithm?

Don't let the name scare you, it's really simple. In essence, an algorithm is a set of instructions, usually given to a computer. Here is an example:

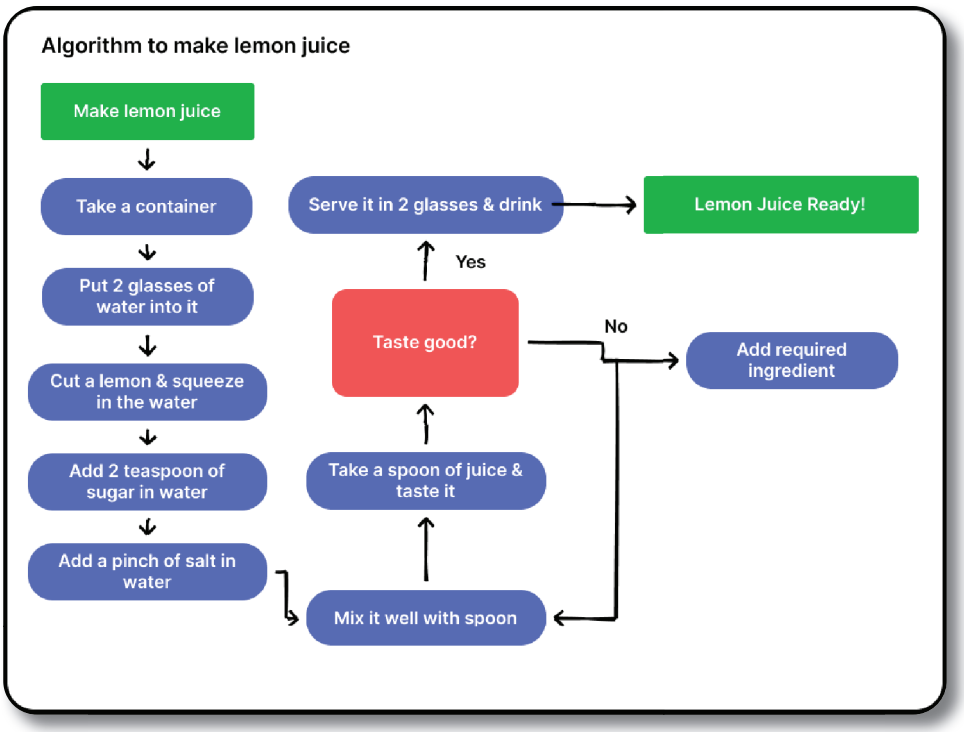
Imagine you have a toy robot that can do anything, and since your room is always messy you want to teach that robot to clean up your toys every day before you come home from school. To program this robot, you just give it a set of instructions. So you give your robot these instructions:

1. Start at the toy box.
2. Pick up one toy at a time.
3. Put each toy in the toy box.
4. Repeat until no more toys are on the floor.

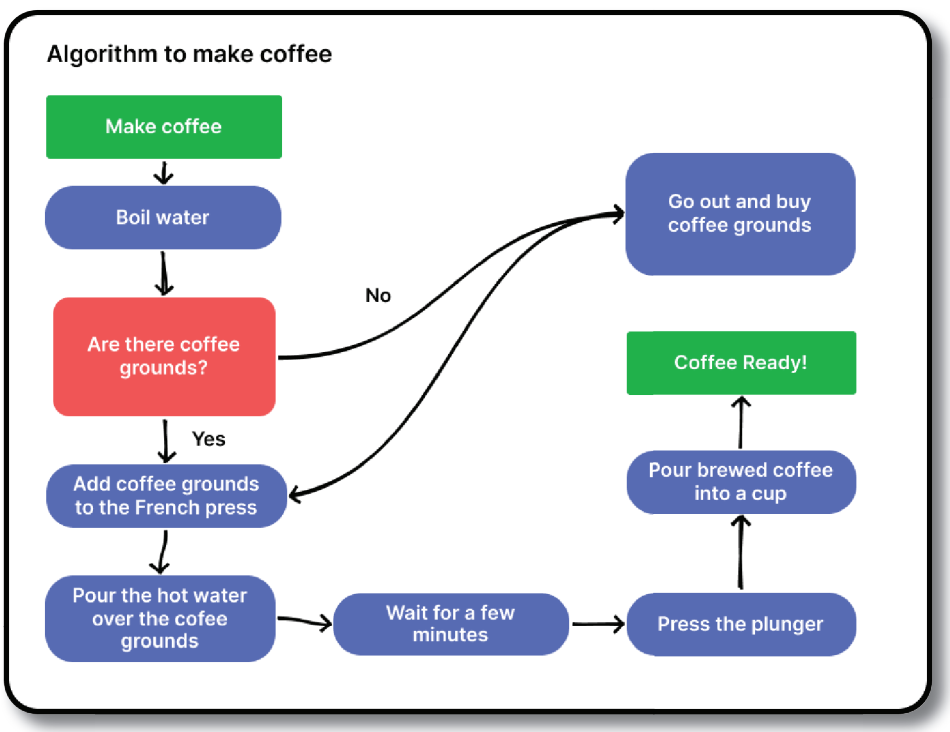
And since the robot is really well made, it does every step in order, and your mom is never mad at you again for having a messy room full of toys on the ground.

The set instructions above is an example of an algorithm. It is a step-by-step guide for the robot to clean up your room.

An algorithm doesn't always have to be about a computer. Here is an algorithm for making lemon juice:



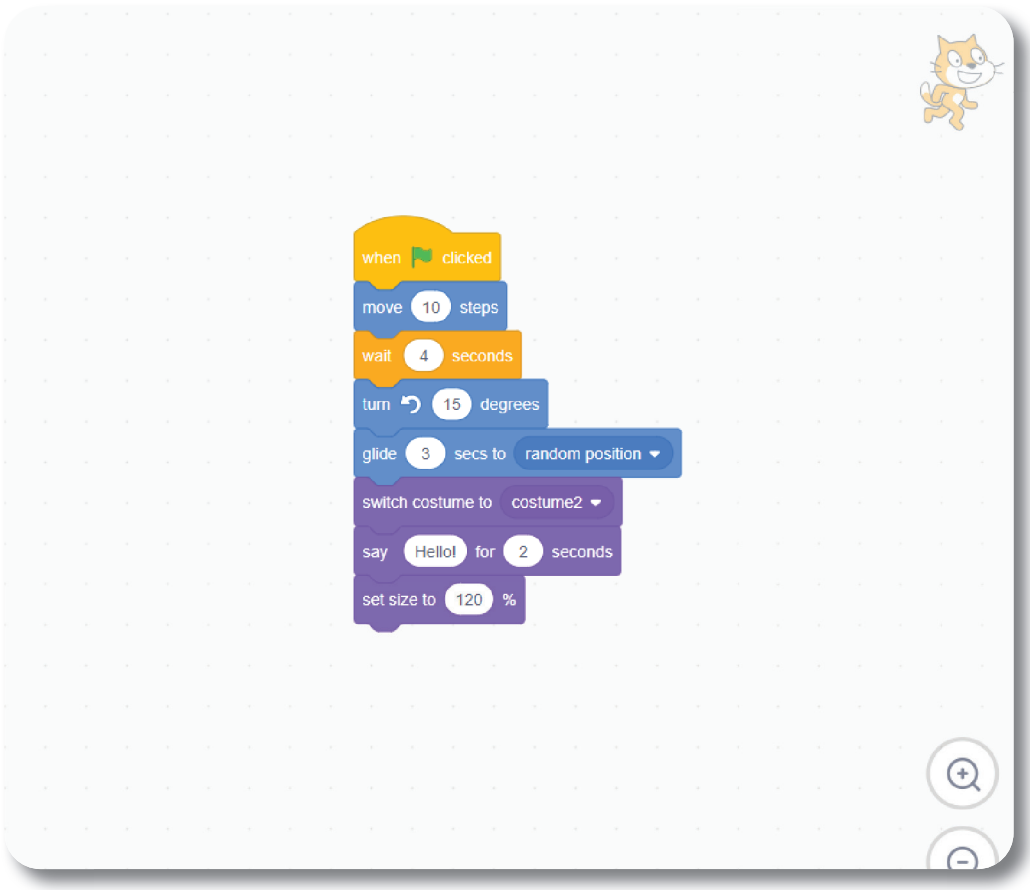
Here is an algorithm for making coffee



Now that we know what algorithms are, let's put our skills to use.

Exercise: On this page, try drawing an algorithm of your own! It can be anything you want.

Here is an example of an algorithm in Scratch. Try to copy this and run it to see what happens!



Quiz

1. Define an algorithm in your own words.

_____

_____

_____

2. Why are algorithms important in programming?

_____

_____

_____

3. Describe the steps of a simple algorithm you encounter in daily life.

_____

_____

_____

4. How do algorithms relate to the programs you create in Scratch?

_____

_____

_____

5. Can you think of a situation where an algorithm might be more efficient than human decision-making?

_____

_____

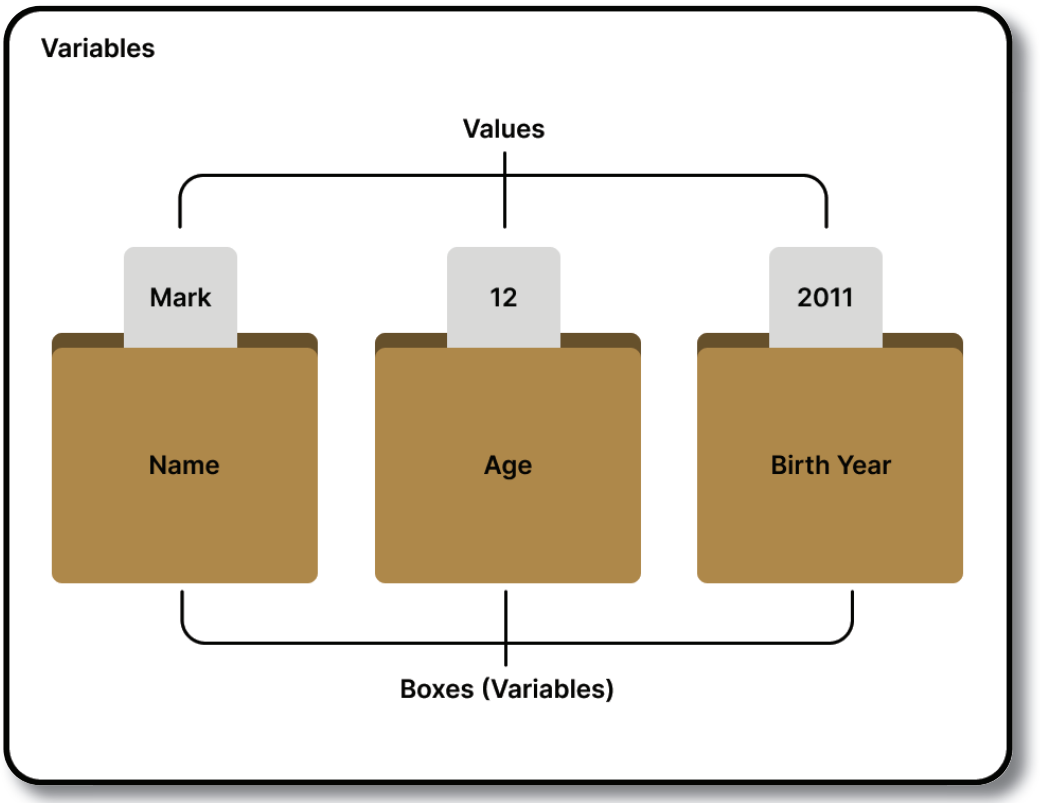
_____

CHAPTER 3:

Basic Coding Concepts

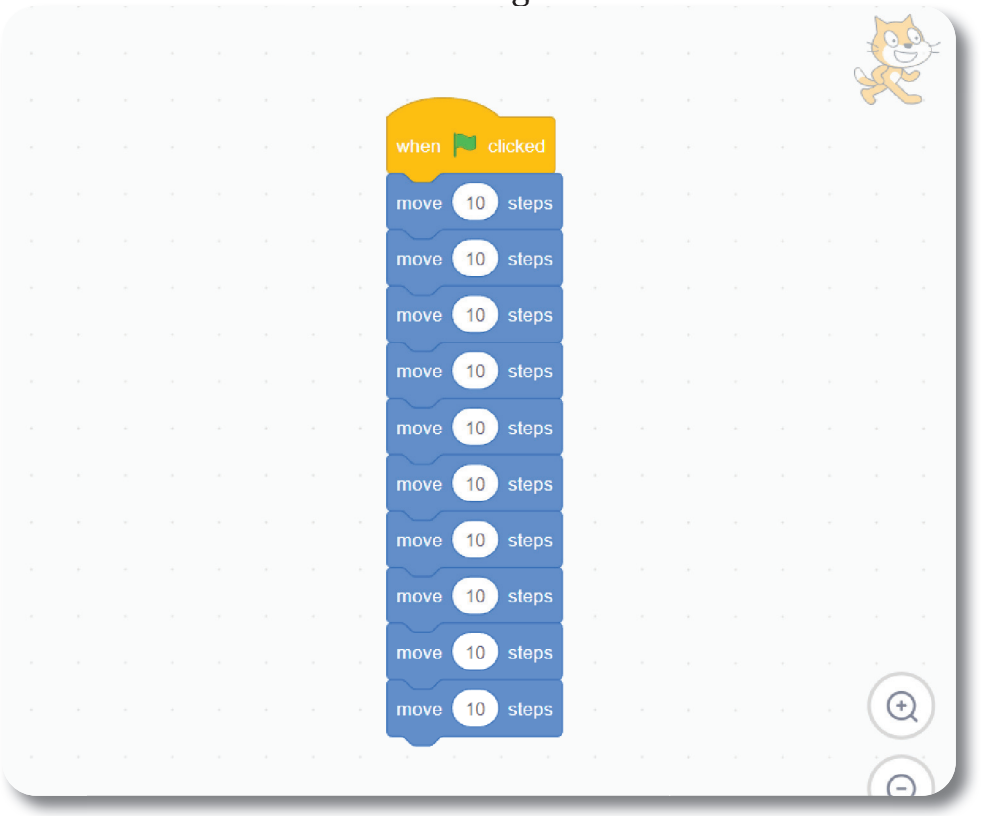
Variables

No matter what programming language you look at, you will come across a thing called variables. You can think of variables as boxes that hold something (values in coding), and that can also be reused. Here is what a set of variables might look like:



Variables are really important in coding because you can use them over and over again.

Imagine you are making a simple game where when the player starts, the Scratch Cat moves 10 steps 10 times. That might look something like this:



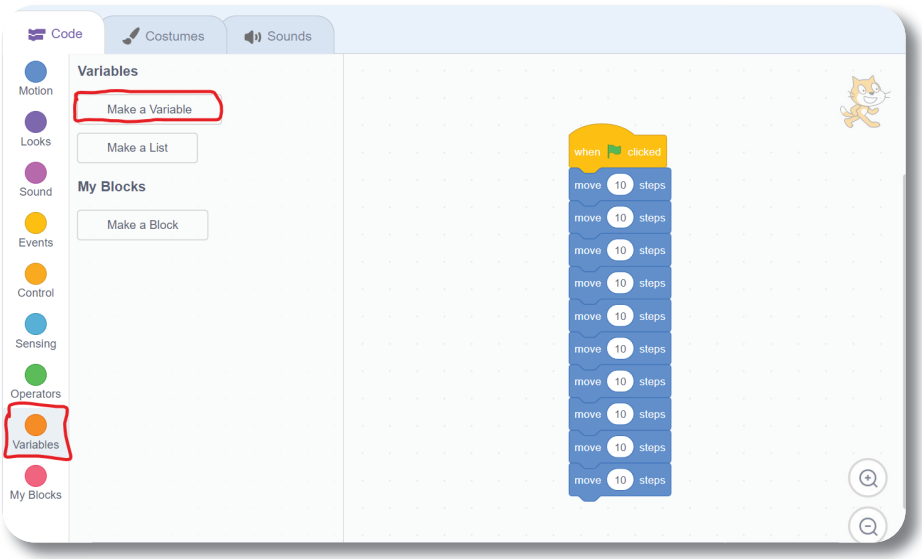
But, after some more development of your game, you decide that 10 steps is too much, and you want your cat to move 1 step at a time. With your current code, you'd have to change every single place you used 10 with 1. What if you decide to change it again? What if you use that value in more places instead of 10 places?

The better thing to do would be to make a variable called "catSpeed", and use it across your code. This way, if you change out the speed, you don't have to go in and change every single place you use it. Let's implement it!

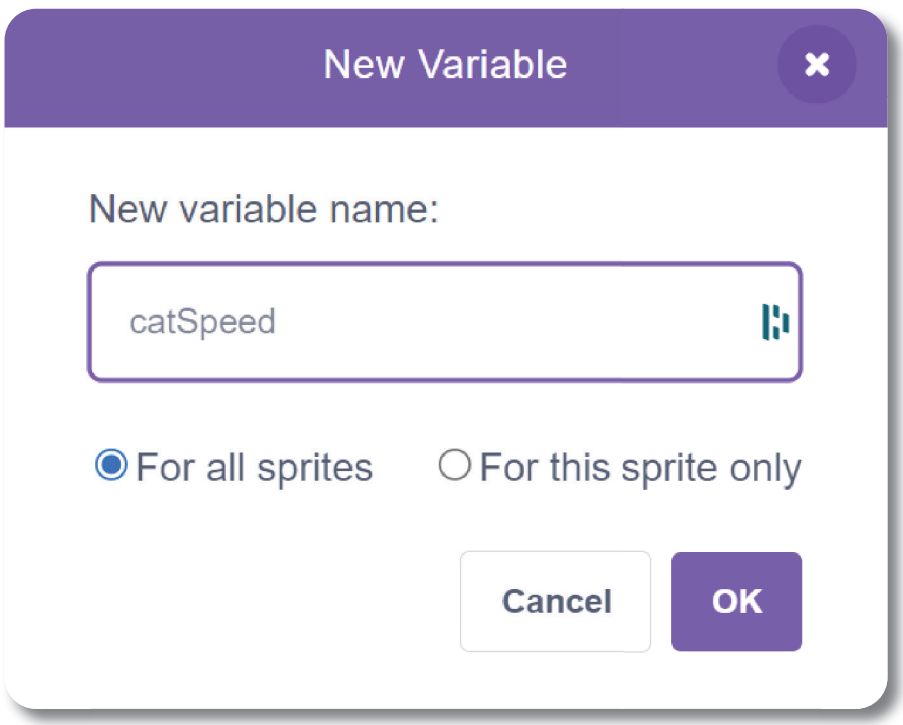
Now, you don't have to follow along here, but I highly encourage you to do so. Before moving on, recreate the above picture on your Scratch Code Space.

To create a variable in Scratch:

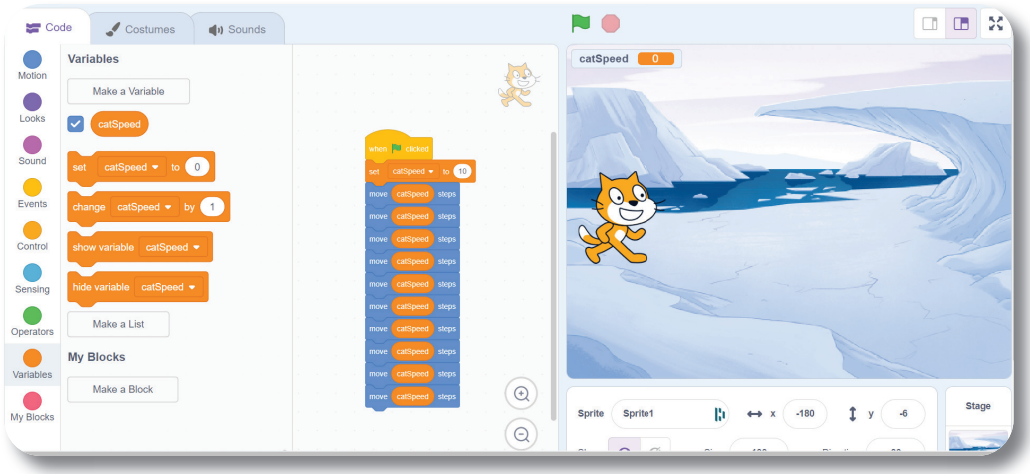
1. Go over to the “Variables” section and click on “Make a Variable”



2. Name your variable “catSpeed” and click “OK”



3. Drag “catSpeed” to where you use 10’s. Also, drag the “set [variable] to [value]” block to the very start of your algorithm, and set the value to 10.



Great job! Now, whenever you have to change the cat speed, you don’t have to do it 10 times. With the power of variables, you can just do it once.

You might have realized that a little box appeared on the top left side of your game view (the right side where the cat is). You can also change this variable there.

Now, variables can hold a variety of different things, like numbers (integers), sentences and words (strings), and true/false statements (booleans)

Great job with this section, let’s move on to some basic data types.

Integers

Integers are just whole numbers, like 10, 300, 1, 2, 5, and 1305. For example, the “catSpeed” variable that we used in the previous section is a variable that holds the type “integer”.

Also in the previous section, the "Age" and the "Birth Date" variables that are in the illustration are also integer variables.

Stuff like 1.5, 203.2, and 67.9 are NOT integers, since they are not whole numbers.

If you have trouble understanding these, please bring this book to your math teacher and ask them to explain this concept.

Strings

Strings are just a set of letters or characters. A word can be a string, a letter can be a string, a paragraph can be a string, a number can be a string, and even a full book can be a string!

In most programming languages, you can write strings by wrapping text with double (") or single (') quotes.

- "103" is a string but 103 is an integer
- "Nice job!" is a string
- "Let's go play some football" is a string
- "12" is a string but 12 is an integer
- "85" is a string but 85 is an integer

In Scratch, however, you don't have to specifically put your string in double-quotes. If something isn't meant to be a string, it doesn't let you enter a string.

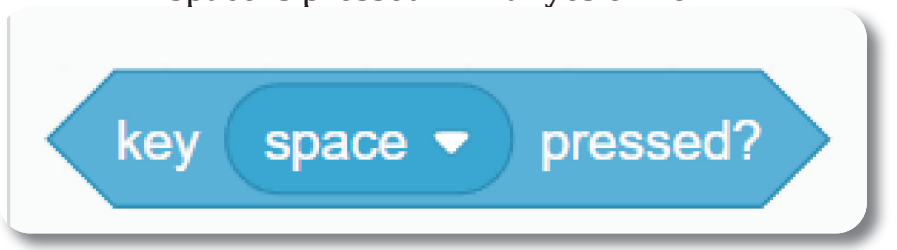
Booleans

Booleans are just "true" or "false" statements. That's it. A boolean is either true or false.

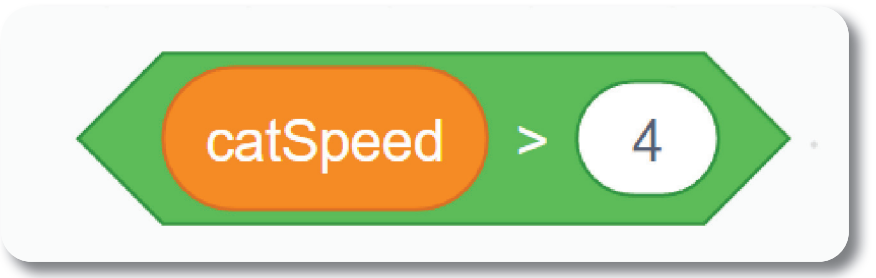
Booleans are usually used in if-else statements, which I will talk about in more detail in Chapter 5.

Here are a few examples of booleans in Scratch:

If "space" is pressed, it is true. If not, it is false.
Think of it as if the computer is answering the question "key space is pressed?" with yes or no.



If the variable "catSpeed" is greater than 4,
it is true. If not it is false.

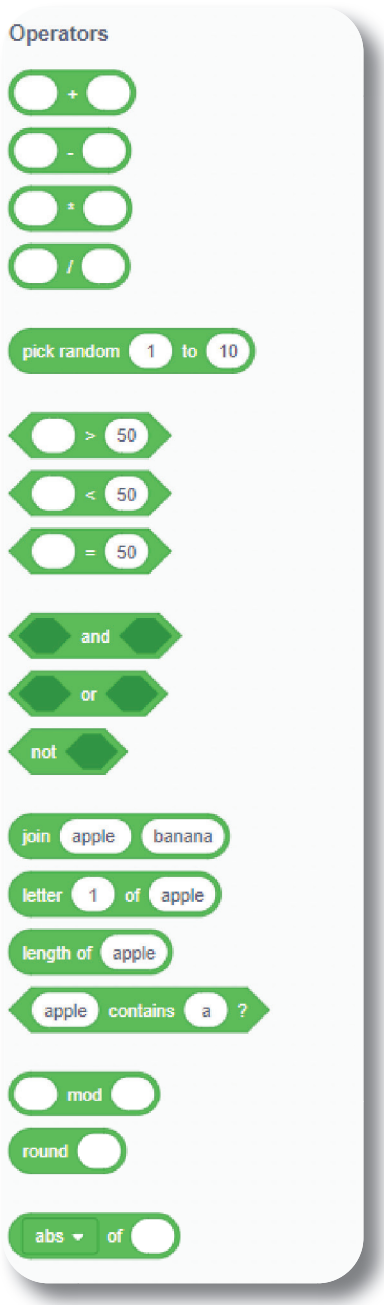


Arithmetic and Logic Operations

Arithmetic operation is just the fancy word for math calculation. For example, adding is an arithmetic operation. Multiplying is an arithmetic operation. Subtraction is an arithmetic operation. Division is an arithmetic operation. Here is the full list:

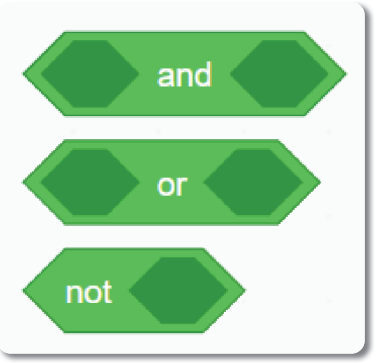
- Multiplication
- Division
- Subtraction
- Addition
- Comparison
- Logical operators

You can find all of these on the “Operators” block section on Scratch



Going back to the previous section on booleans, you can see the “comparison” operation in action where it compares “catSpeed” to the number 4.

You already know about arithmetic operators from your math class, so instead, let's talk about logical operators. There are 3 logical operators in Scratch: and, or, and not.



Let's call the "and" operator Andy. For Andy, everything has to be perfect. Andy, put simply, is a perfectionist, so even one "false" can make it "false".

Here is a table showing different outcomes for the operator "and" where "apples" and "peaches" are different variables.

The "and" operator

apples	peaches	apples and peaches
true	true	true
true	false	false
false	true	false
false	false	false

Let's call the "or" operator Orion. Orion is more flexible than Andy. For Orison, it's enough for one of the values to be true.

The only thing Orison doesn't accept is both values being false, which in my opinion, is pretty reasonable. Here are the different values for Orison:

The "or" operator

apples	peaches	apples or peaches
true	true	true
true	false	true
false	true	true
false	false	false

Let's call the "not" operator Notara. Notara just flips things around. If you have a variable "apple" and it is "true", "not apple" will be false. If apple is false, then "not apple" is going to be true.

Your friends Andy (and), Orion (or), and Notara (not) are going to be very useful in the future. For now, let's move on to input and output.

Input and Output

Input and output might sound a little intimidating but don't worry, it is really simple to understand.

Input is like giving your friend some information or asking a question. For example, if you tell your friend, "What's the weather like today?" or "Can you draw a picture of a cat?" – you're giving them input.

Now, your friend is really good at understanding things and doing stuff. So, after you give them input, they do something with that information. That's where output comes in.

Output is like your friend's response or what they do for you based on the information you gave. If you asked about the weather, they might say, "It's sunny today!" or if you asked them to draw a cat, they might show you a picture of a cute cat.

So, in simple terms, input is what you tell your friend, and output is what your friend does or says in response. It's like a conversation between you and your super-smart friend!

In the computer world, input is someone giving some information or asking a question to the computer (instead of your friend), and the computer producing a response for that.

For example, when you are playing a video game and you press "W" to walk forward, you are giving the computer an "input", and if the game works correctly and you walk forward in the video game, that is the output.

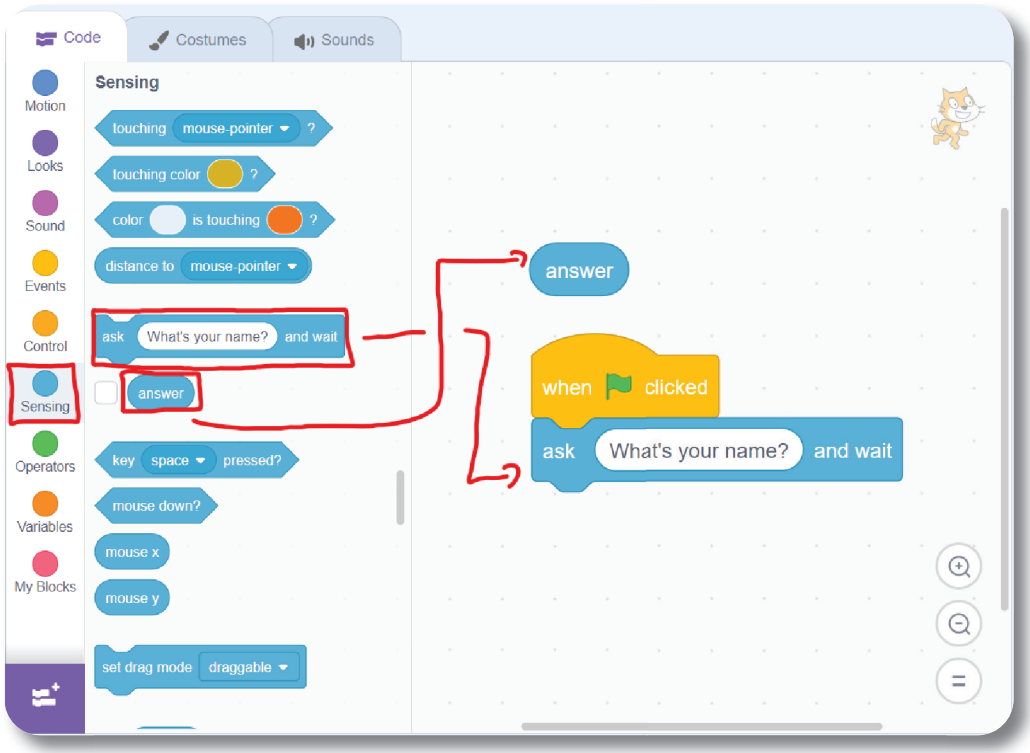
When you want to Google something and write your question to the Google search bar; you pressing the keys "Are pandas cute?" in order is the input, and that sentence showing up in the search bar is the output. You pressing enter is the input, and Google doing the search is the output.

When you want to put a Scratch code block on the code area and click on an "if" block (to hold it), that is the input. You holding it is the output.

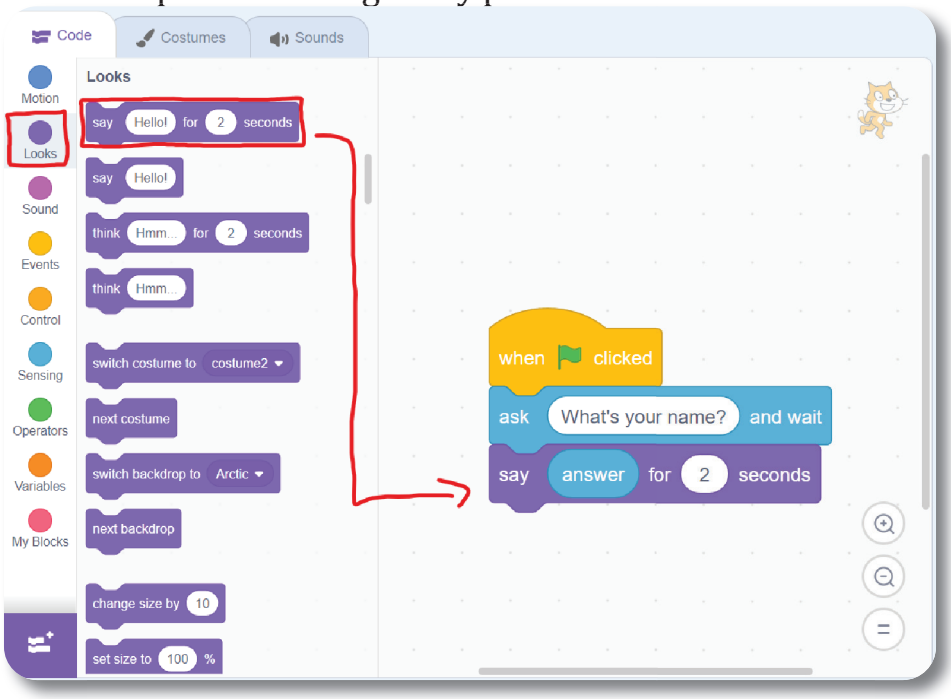
So, how does this work in practice? Well, let's do some input-outputting in Scratch!

First, go over to the "Sensing" section on the left sidebar, and put a the "ask _____ and wait" block under the "when [green flag] is

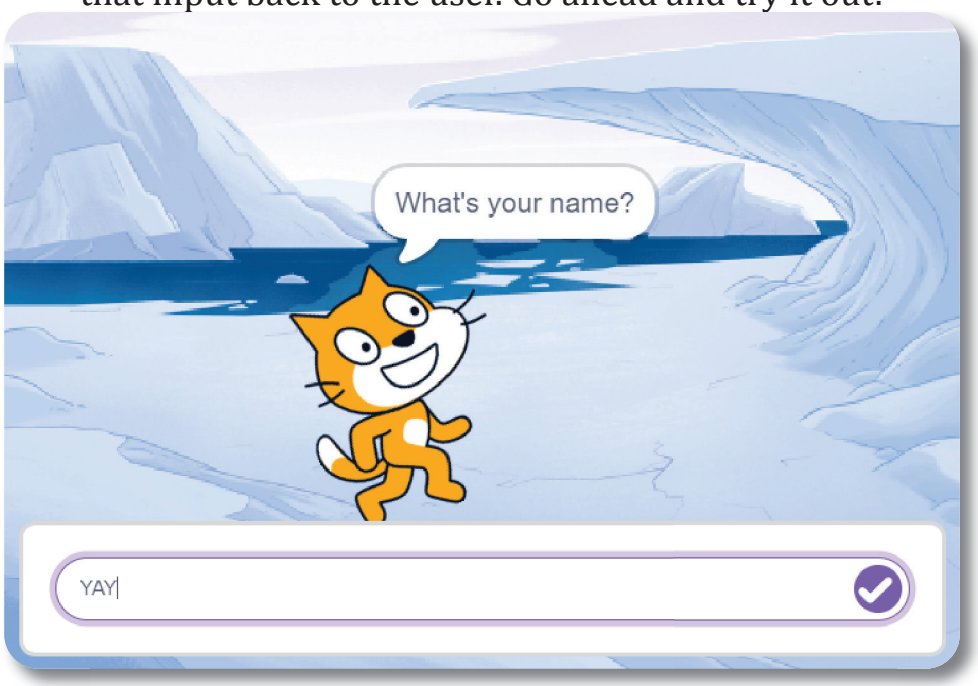
clicked" block. Then, put the "answer" variable that is located right under the "ask ___ and wait" block and put it somewhere in the screen, we will use this variable later. The "ask" block will take input from the user, and after the user gives the input, the "answer" variable will hold that input. We can use this "answer" variable to do some cool interactive stuff"



Now, go over to the “Looks” section and put a “say ___ for ___ seconds”. Replace the thing to say part with our “answer” variable.



Now, what this code will do is take input from the user, and say that input back to the user. Go ahead and try it out!



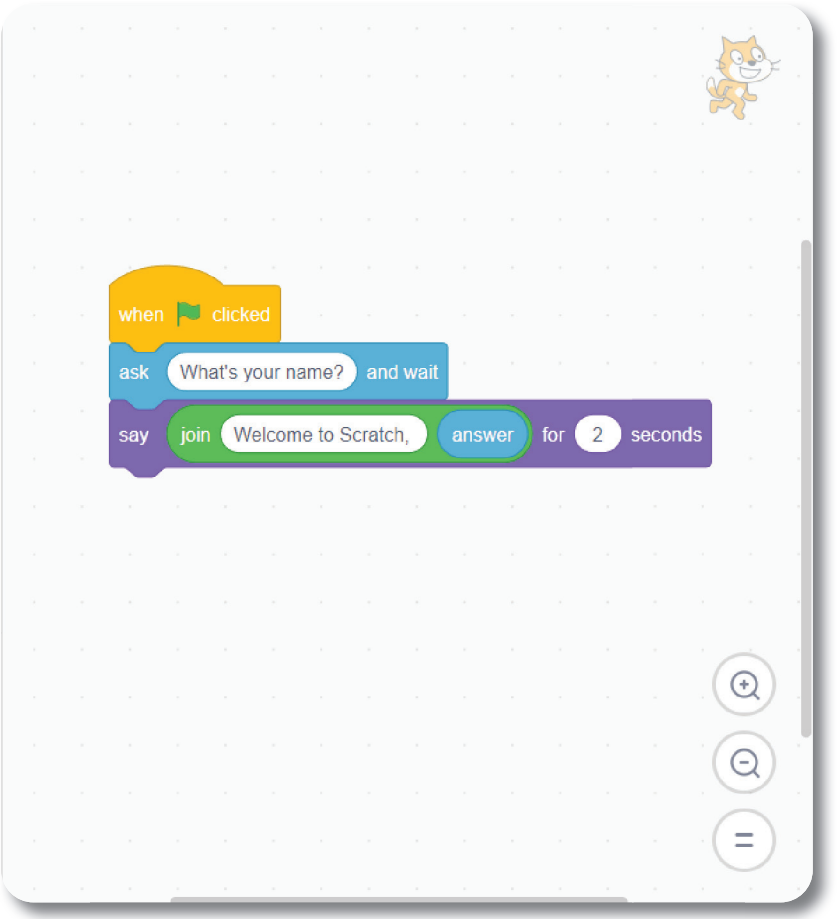
Now, we can customize this even further, using arithmetic operations! The cool thing about coding is arithmetic operations don't always have to be about numbers. For example, you can use the "+" operation to add 2 strings. The fancy term for this is "string concatenation"

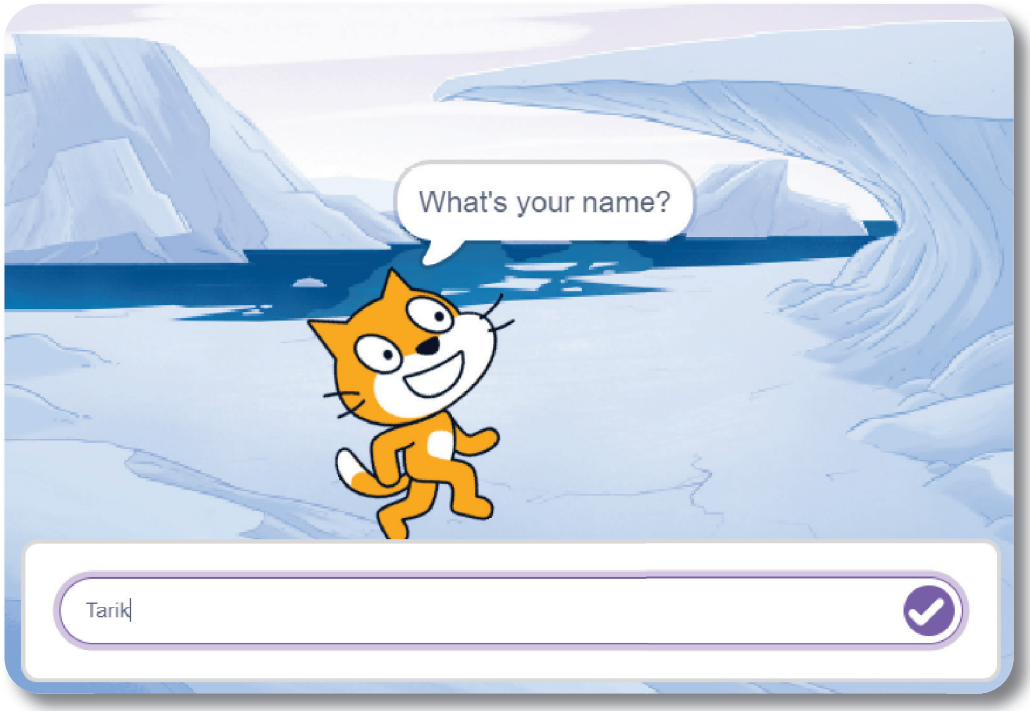
Adding 2 strings

```
"Hello" + " Josh" = "Hello Josh"
```

This is how it works in most programming languages, but in Scratch, there is a custom block for this under the Operators section.

We can use this to add “Welcome to Scratch,” and our “answer” variable to produce a custom output. So if the user entered in “Tarik” as the input, the Scratch Cat would say “Welcome to Scratch, Tarik”. Note that Before looking at the solution, I encourage you to try to do this yourself.





Great, now you know the basics of input-output, we can move on to some more fun stuff. Wait ahead, loops and conditionals!

Quiz

1. What are variables and why are they important in coding?

_____

_____

_____

2. Explain the difference between integers, strings, and booleans.

_____

_____

_____

3. How do arithmetic and logic operations work in Scratch?

_____

_____

_____

4. Give an example of a real-world problem that could be solved using basic programming concepts.

_____

_____

_____

5. Why is understanding data types crucial in programming?

_____

_____

_____

6. Can you create a simple Scratch program using variables, data types, and basic operations?

_____

_____

_____

CHAPTER 4:

Loops and Repetition

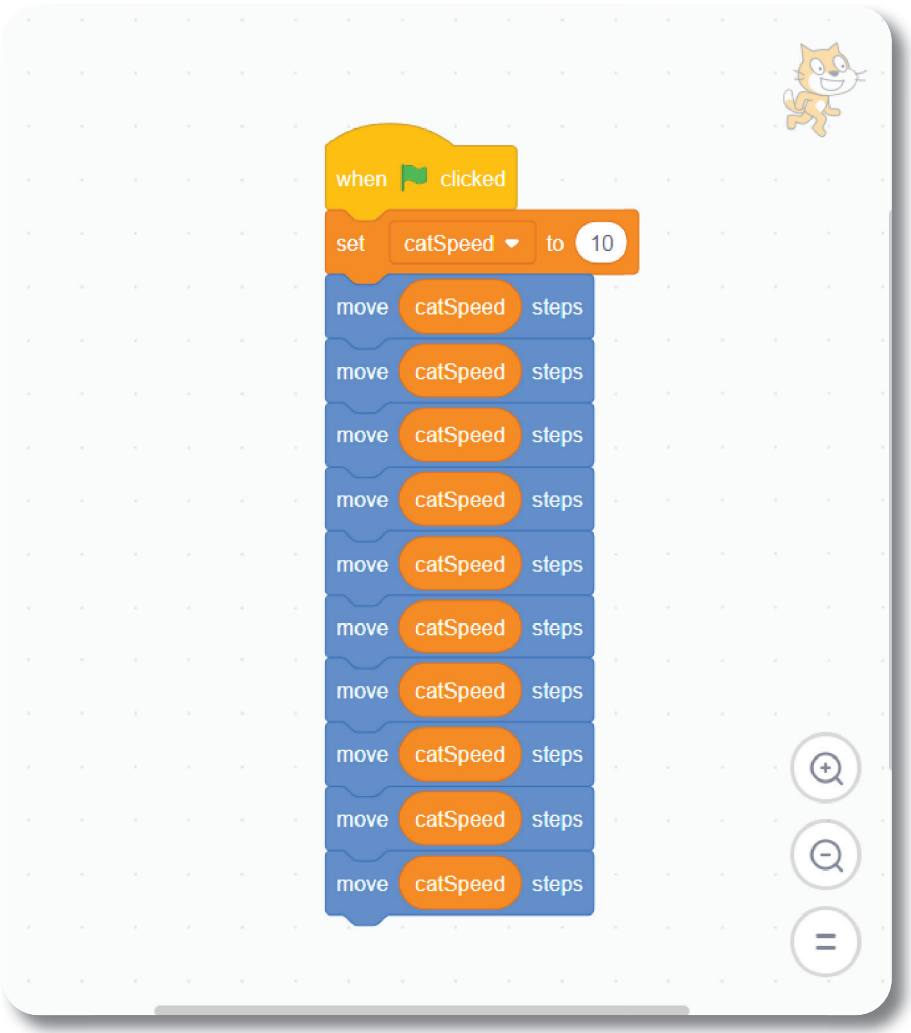
Understanding Loops

Loops and repetition go hand in hand. Loops are your way of telling the computer "do something over and over again". So for example, if you want your computer to say "Hey User!" 1000 times, without loops, you'd have to tell it to say "Hey User!" 1000 separate times. But with loops, you can tell the computer to "Say Hey User! 1000 times" once.

I know this may be confusing at first, but bear with me. Let's say you bought more toys since the last time you made your toy cleaning robot, so you want to update it. But this time, you also want it to pick up each toy and say "Hi" to it. Instead of saying, "Pick up the dinosaur toy, say 'Hi.' Pick up the robot toy, say 'Hi,'" and so on, you can tell the robot, "Pick up each toy and say 'Hi' until there are no more toys." The robot will keep doing this until all the toys are picked up.

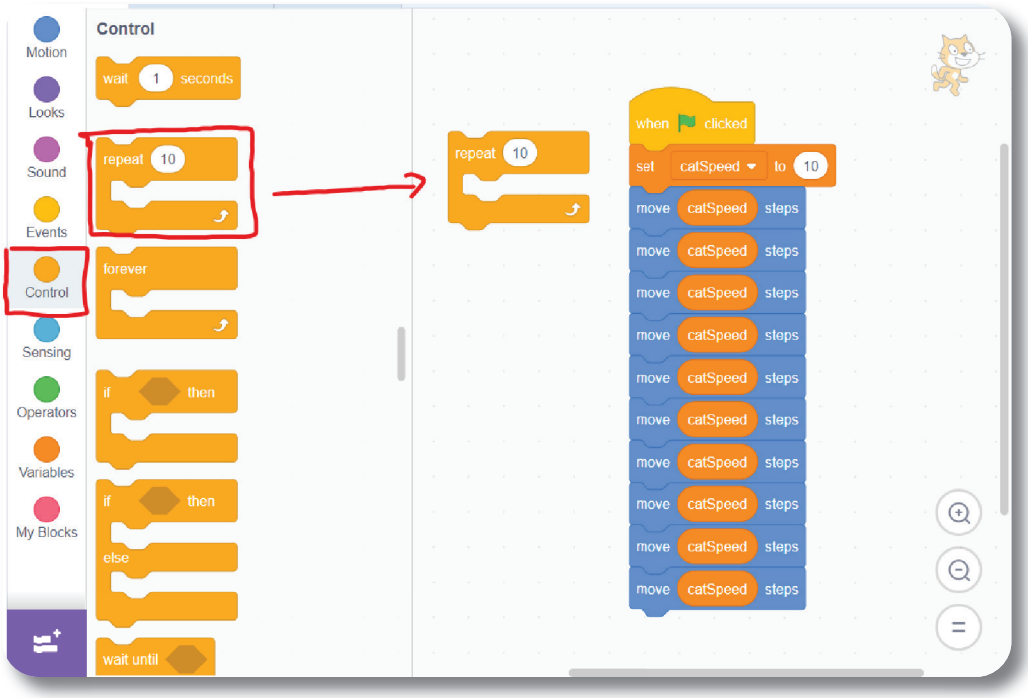
Loops in Scratch

Let's see how this works in Scratch. Remember this algorithm from the previous chapter's "Variables" section?

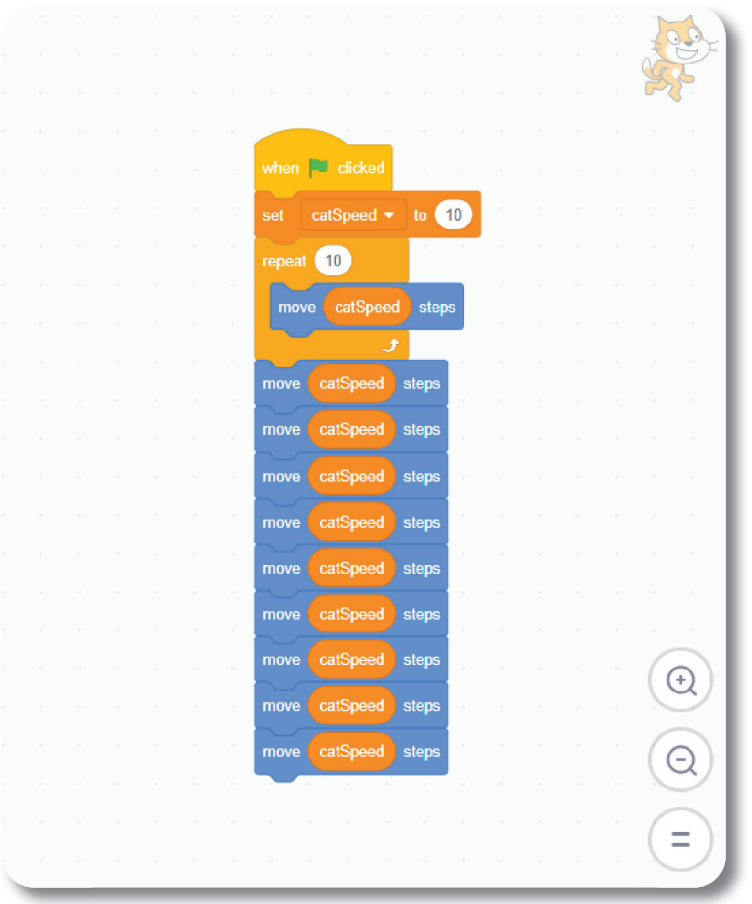


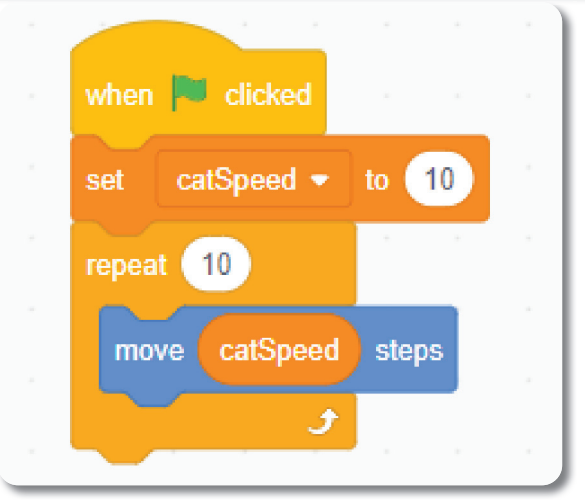
Well, hopefully by now you can see how repetitive this code is. It is not efficient for us to say the same thing over and over again. This is the perfect place to use loops.

Loops are located in the "Control" section. Grab a "repeat" block and drop it into the code area.



This block allows us to repeat something as many times as we want, 10 in this case. The block under it repeats something forever, but we won't need it for now. Now, grab one of the "move catSpeed steps" blocks and put it inside the repeat block. After that, put that repeat block under the "set catSpeed to ___" block. You can delete the remaining blocks (by right-clicking on them and selecting "Delete blocks"), since we don't need them anymore.





Now, if you run the code, you will see that the same thing happens. The difference is that now, it's easier to read our code, almost like plain English: "Repeat move catSpeed steps 10 times". Not quite plain English, but still pretty readable.

Quiz

1. What is the purpose of a loop in programming?

_____

_____

_____

2. Can you design a loop that would solve a repetitive task you do daily?

_____

_____

_____

3. Discuss how loops can make coding more efficient.

_____

_____

_____

4. Create a simple loop in Scratch and explain what it does.

_____

_____

_____

5. Why is it important to understand the concept of repetition in coding?

_____

_____

_____

6. Give an example of a scenario where a loop might not be the best solution.

_____

_____

_____

CHAPTER 5:

Making Decisions With Conditionals

Introduction to Conditionals

Conditionals allows us to do stuff if something happens.

Let's say you have your really smart friend, Fluffy.

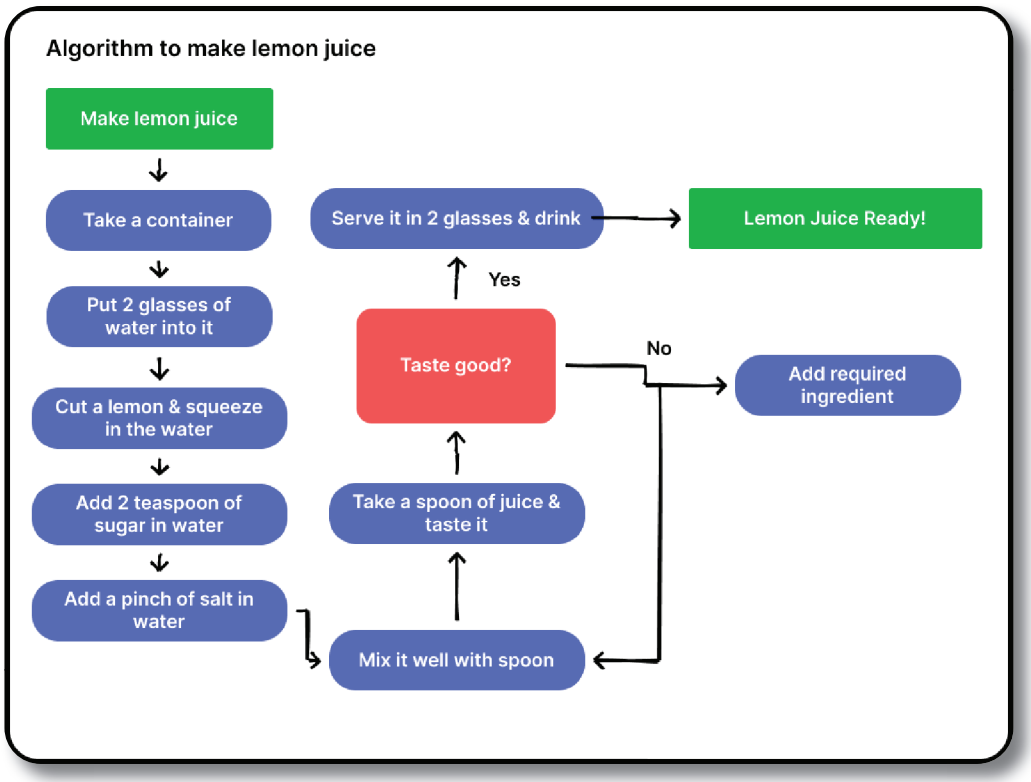
Imagine you say, "Fluffy, if it's raining, grab your umbrella; if it's sunny, wear your sunglasses." This is like a conditional instruction!

In coding, we use conditionals to tell the computer what to do based on certain conditions. It's a bit like giving instructions to Fluffy. For example, you might tell the computer, "If it's raining, show a picture of an umbrella; if it's sunny, show a picture of sunglasses."

Let's say you want it to do something. You can say, "Fluffy, if I say 'magic word,' do a happy dance; if I don't say the magic word, just sit quietly."

In coding, you might have a program that says, "If the user enters the correct password, show them their messages; if they don't enter the correct password, tell them it's wrong."

Let's bring back the lemon juice making algorithm from Chapter 2.



Try to find the conditional here.

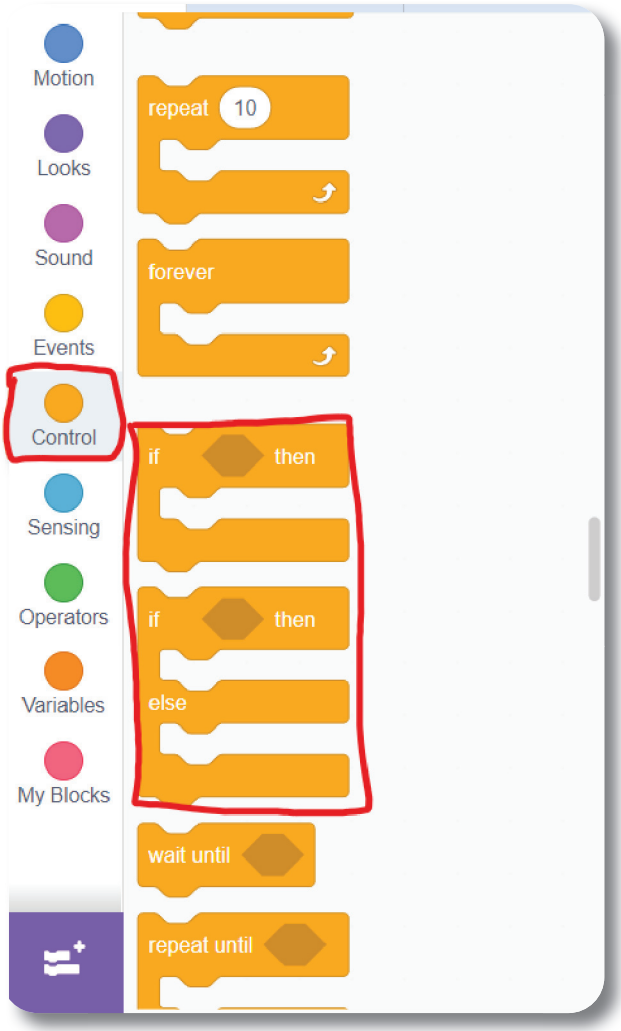
Great job! It is the red "Taste good?" block.

If it tastes good, **then** you serve it

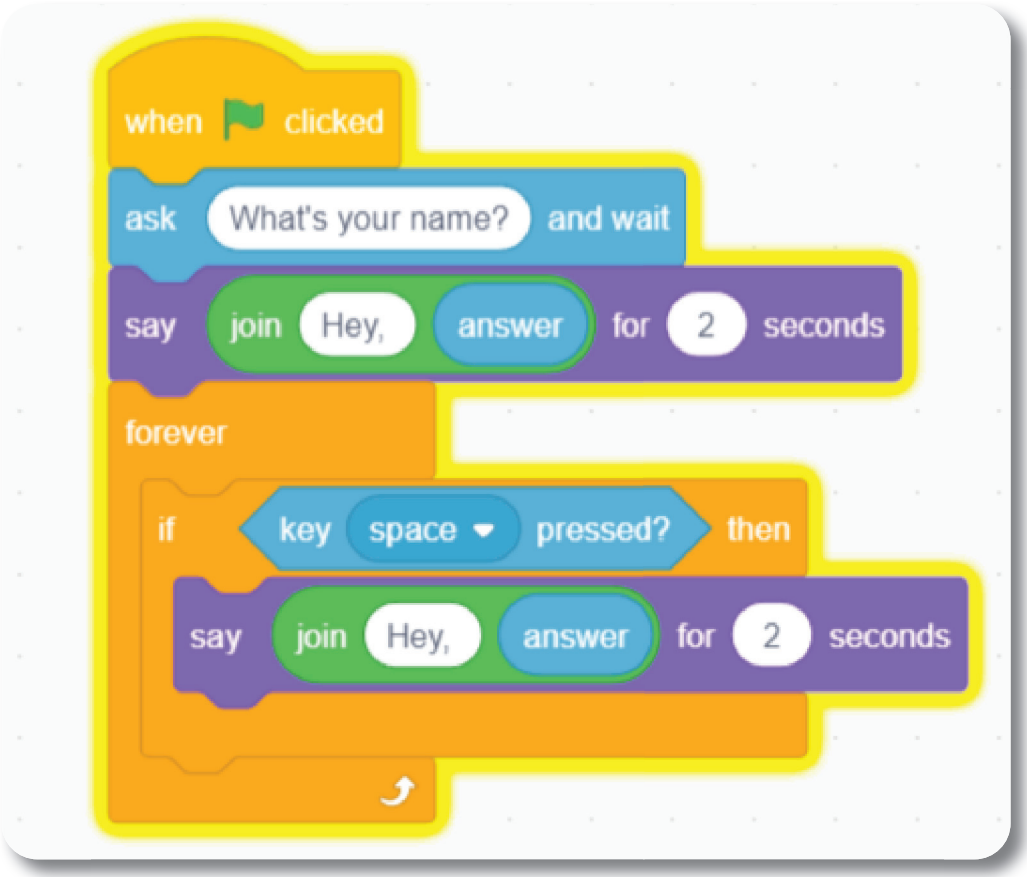
Else, you add the required ingredient.

Conditionals in Scratch

In Scratch, conditionals are in the same place as loops are: Control section.



Here are a few examples of code that use these "if _ then ___" blocks:



This algorithm here goes like this:

1. Ask "What is your name?" and wait for an answer
2. When you get the answer, say "Hey, [their answer]" for 2 seconds
3. Enter into a "forever loop"
4. If the space key is pressed, just say "Hey, [their answer]" for 2 seconds.

You might be wondering why we have the "forever loop" there. The reason is that we want to continuously keep checking if the user has pressed the button. If we didn't have the loop, our code wouldn't check for key presses forever, and it would check for it only once.

Nice job with this section, get ready to learn all about functions!

Quiz

1. What are conditionals and how are they used in programming?

_____

_____

_____

2. Create a simple conditional statement and explain how it works.

_____

_____

_____

3. How do if-else statements enhance the functionality of a program?

_____

_____

_____

4. Can you think of a real-world decision-making process that resembles a conditional statement in coding?

_____

_____

_____

5. Why is it important for a programmer to understand conditionals?

_____

_____

_____

6. Give an example of a Scratch program that uses conditionals.

_____

_____

_____

7. How do conditionals contribute to the interactivity of a program?

_____

_____

_____

CHAPTER 6:

Functions and Splitting Up Our Code

Intro to Functions

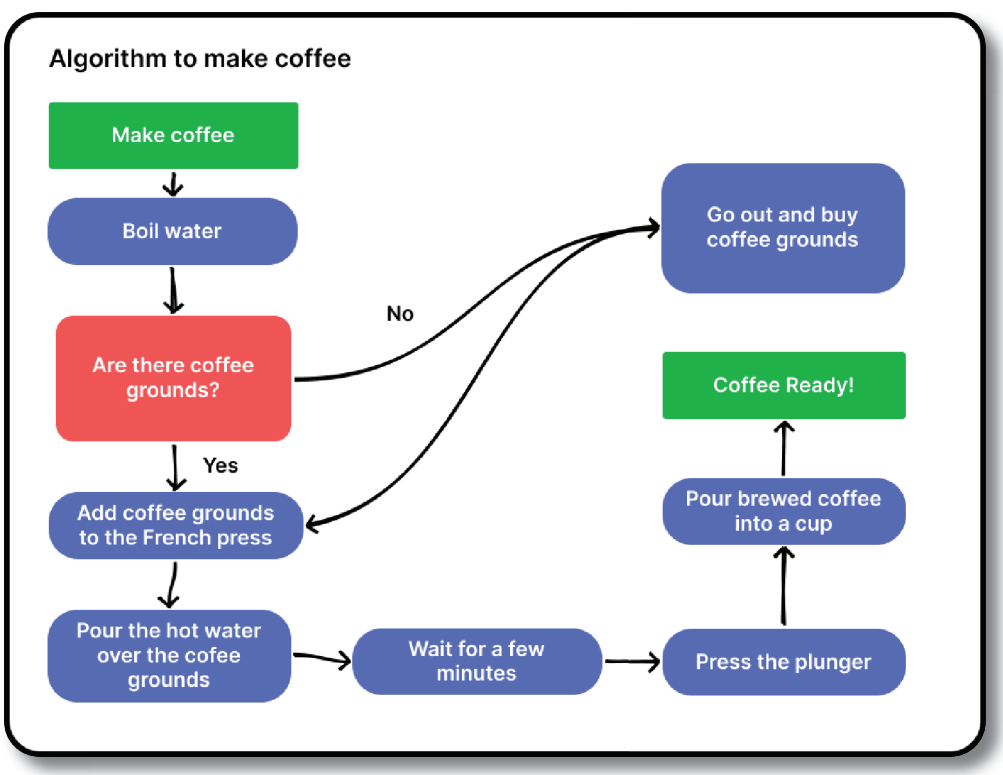
Again, as with everything, don't let the word "function" scare you.

The function is originally a math term that you are going to learn about in high school, but it has made its way into the world of coding throughout the years.

A function at its core is a set of instructions. Sounds familiar? That is exactly what algorithms are!

You can think of functions as bundled-up algorithms that you can use wherever you need in your code. They are variables where instead of storing values, they store algorithms.

Imagine that we have a program where we make coffee all over our huge codebase. First, let's bring back our "coffee-making algorithm" from Chapter 2.

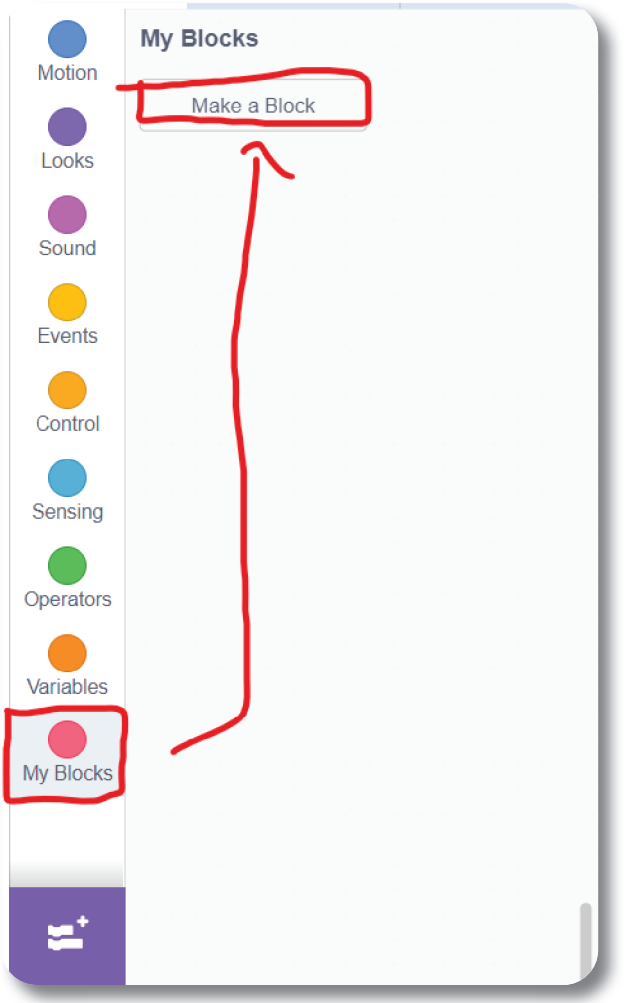


This is a lot of code. Imagine rewriting this all over your code over and over again. It will look ugly and it won't be efficient. What we can do is define a function called "Make coffee" once and just use that for the rest of our code.

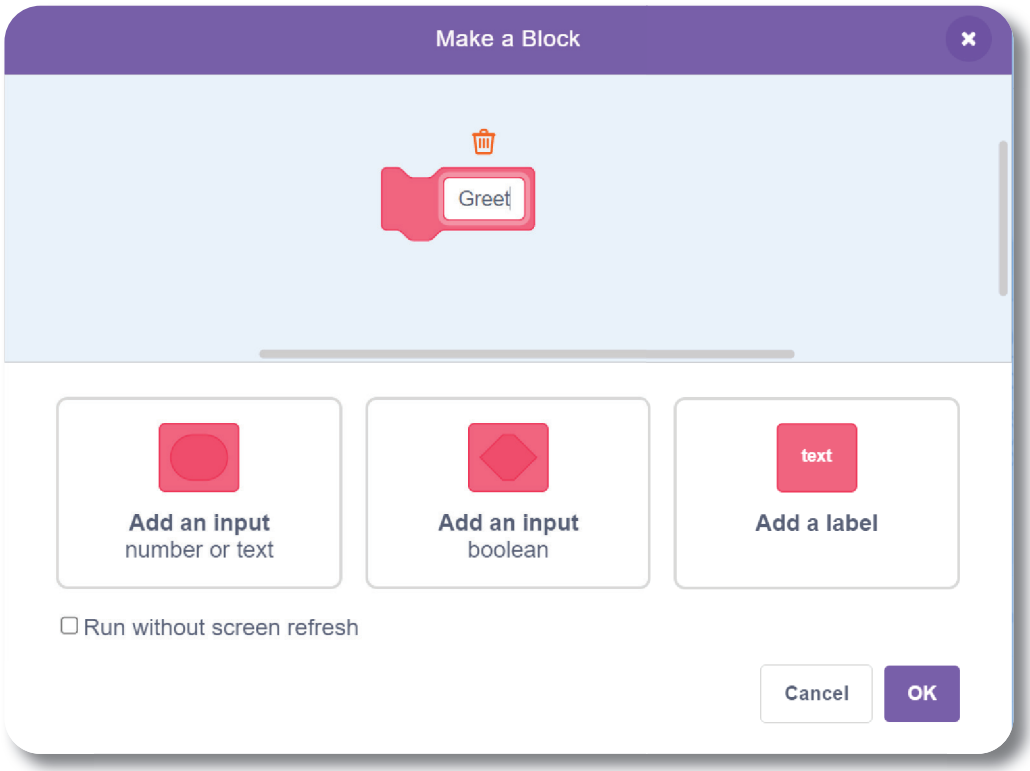
I want you to imagine that we defined that function on the green "Make coffee" block. Now, whenever we use this block, we will call this function, which means no more repetition!

Functions in Scratch

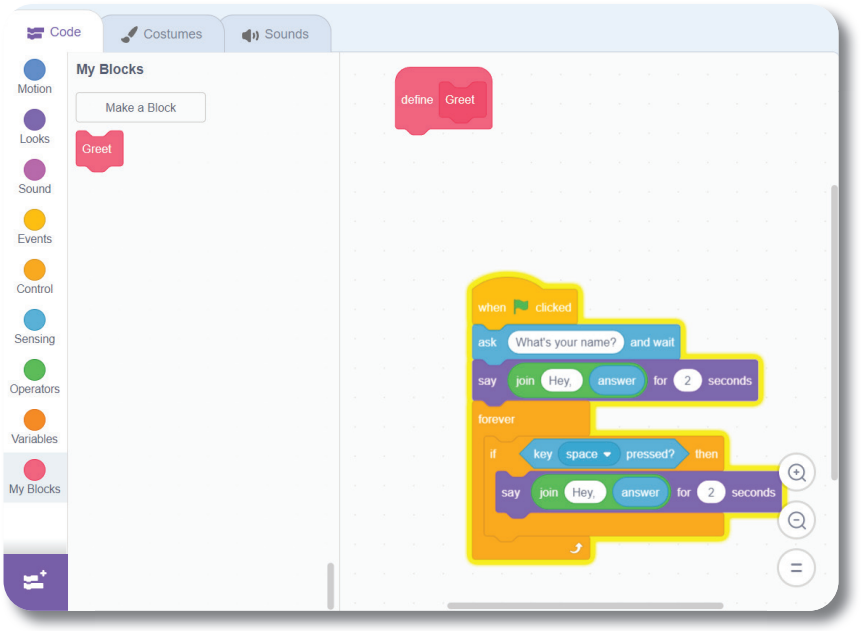
To make or "define" a function in Scratch, you need to go over to the "My Blocks" section and click on "Make a Block",



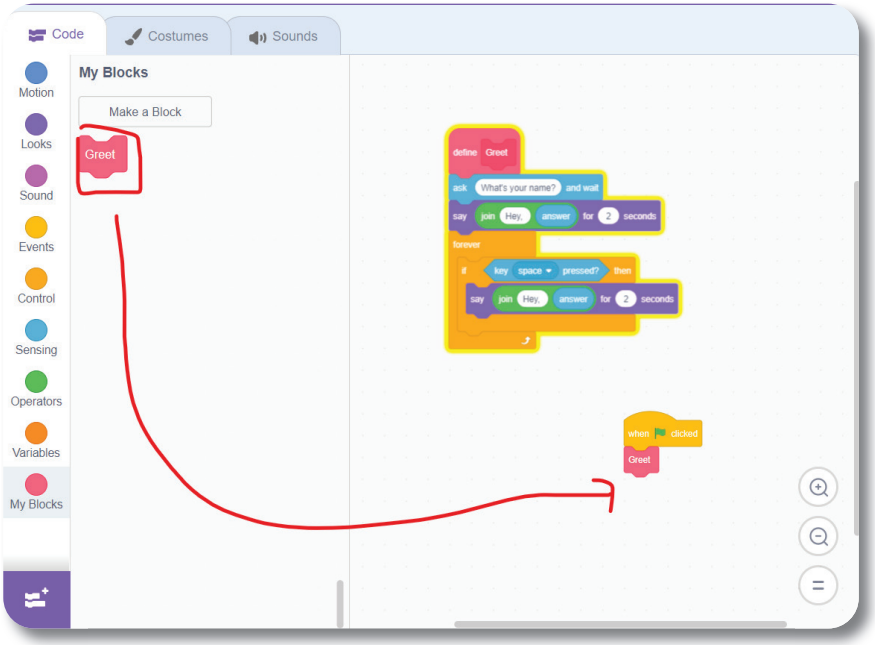
After that, you will be greeted by a pop-up that will ask you what to edit the block. For now, just change the block name to whatever you want. I will be making the last algorithm we made on Chapter 5 into a function so I will call mine "Greet". After you are done, press OK.



Now, your screen should look like the image below (except if entered in a different name than "Greet"). The "define" block allows us to define our function, and the block with the name of your function ("Greet" in my case) is the block that allows you to use that function.



I will move my algorithm over to the “define Greet” block from the [when greenflag clicked icon] block, and I will call or run my function under the [when greenflag clicked icon] block like so



Now, I can reuse this Greet function anywhere I want!

In the next section, we will be putting our skills into use by building a small game called Funbytes!

Quiz

1. Define a function in the context of programming.

_____

_____

_____

2. Why is it beneficial to use functions in your code?

_____

_____

_____

3. Create a simple function in Scratch and explain its purpose.

_____

_____

_____

4. How do functions contribute to the readability and organization of code?

_____

_____

_____

5. Can you think of a daily task that could be simplified into a function-like process?

_____

_____

_____

6. Discuss the importance of organizing code into functions when working on larger projects.

_____

_____

_____

7. How might functions be used to enhance the game or application you are developing in Scratch?

_____

_____

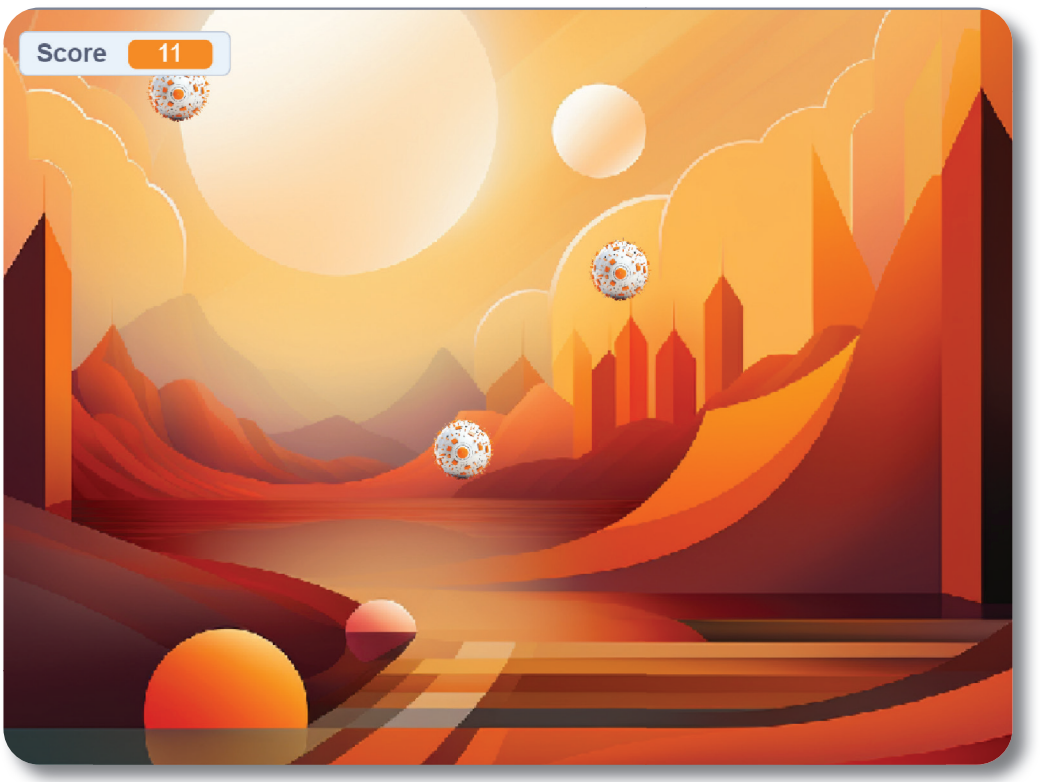
_____

CHAPTER 7:

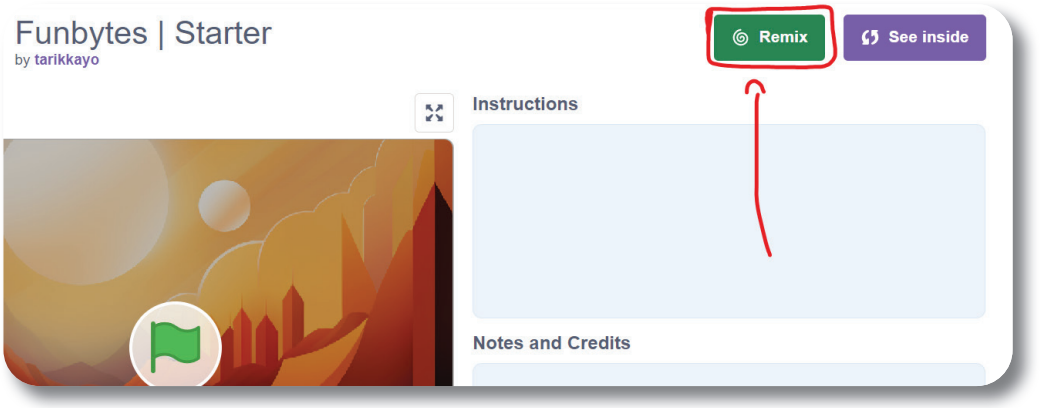
Funbytes!

Now, it's finally the moment you have been waiting for: Funbytes!

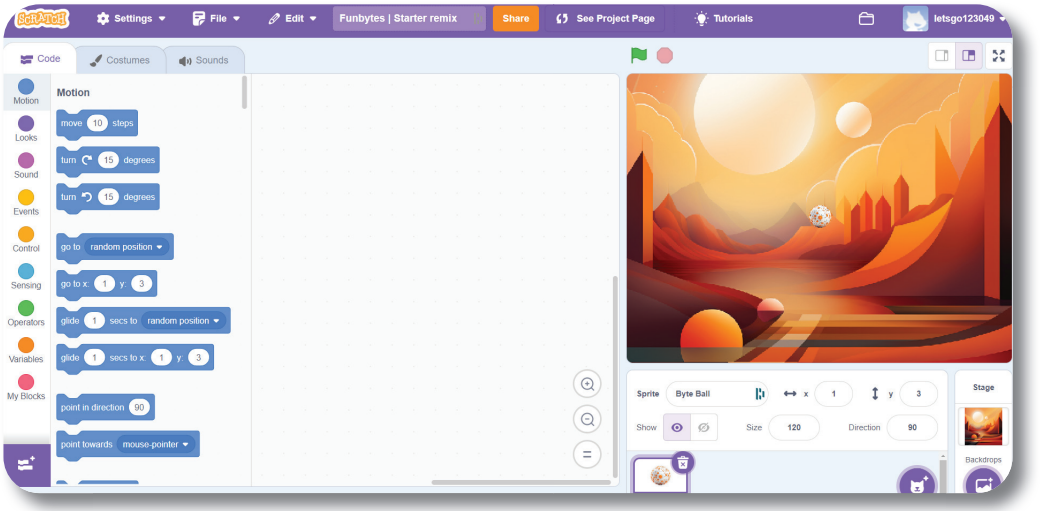
By going through this chapter, you will utilize everything you learned up until this point. We are going to build a byte-catching game called Funbytes. In this game, our player will try to catch bytes with their mouse pointer. Here is what it's going to look like:



To begin, I want you to copy my starter project to yourself on https://scratch.mit.edu/projects/937717627. To do it, just go to the website and press on the green "Remix" button.



Once you do, your screen should look like this:

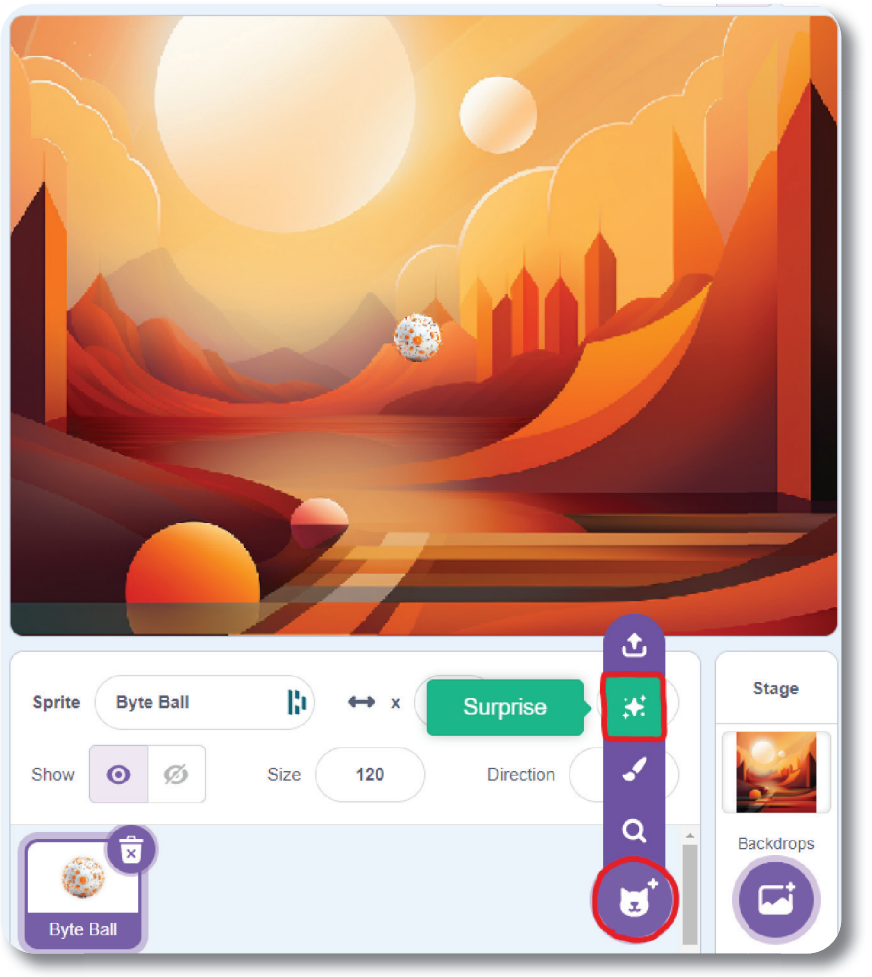


The ball in the middle of the game screen is what I'd like to call the "Byte Ball". It is going to be our user's primary target.

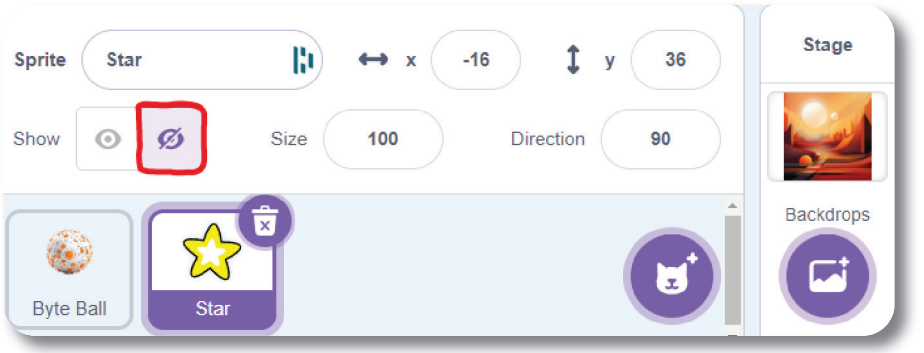
The Game Manager

Before moving forward, we need to create a game manager. The game manager will manage all of our Byte Balls falling. You will see what I mean more clearly as we go on.

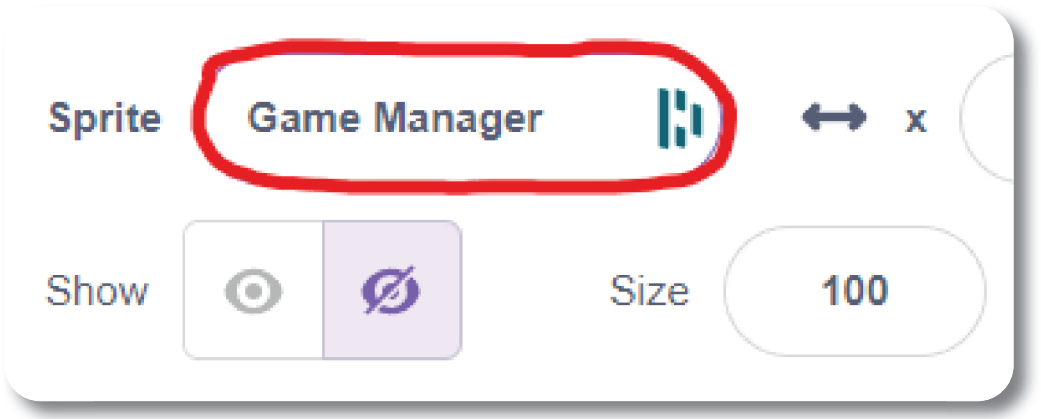
For now, I want you to hover over the purple dog icon and pick the Surprise option (the starry icon). This will give you a random fun sprite that will be our game object.



It gave me a star! Unfortunately, stars aren't a part of the Byteland. This random sprite's purpose is to control our game, so we need to hide it. You can think of it as an invisible game manager. To hide it, just press on the eye with line through it on the "Show" toggle.



Now, just rename the sprite to “Game Manager”, since it’s not clear what our mysterious “Star” is doing by its name.



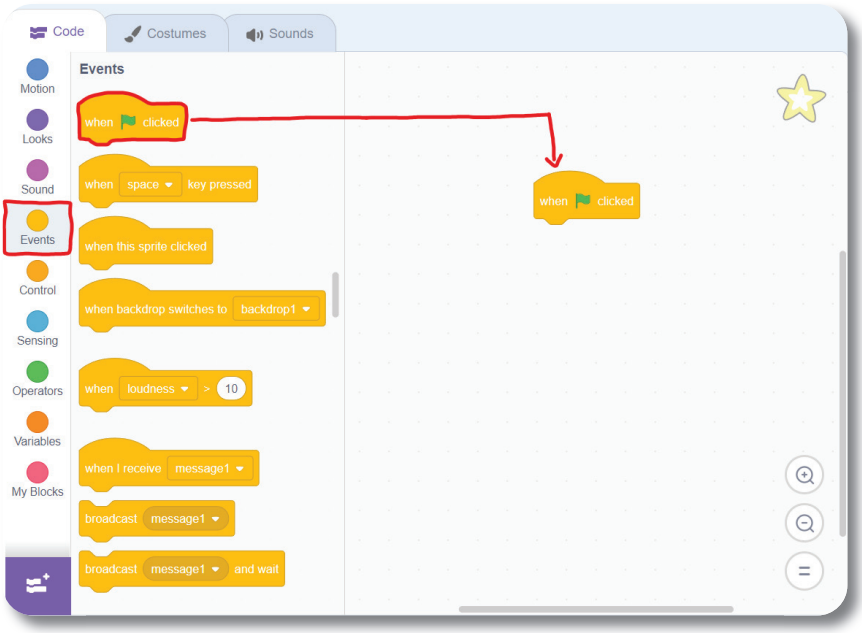
Notice how the “Star” sprite is selected with the purple highlight? This means that whatever changes we make to the code, will be specific to the “Star” sprite. It won’t affect our “Byte Ball”. In fact, our “Byte Ball” will have a special algorithm to make it fall and detect mouse hovers.

Now, we are ready to start coding our game manager! The algorithm will be dead simple.

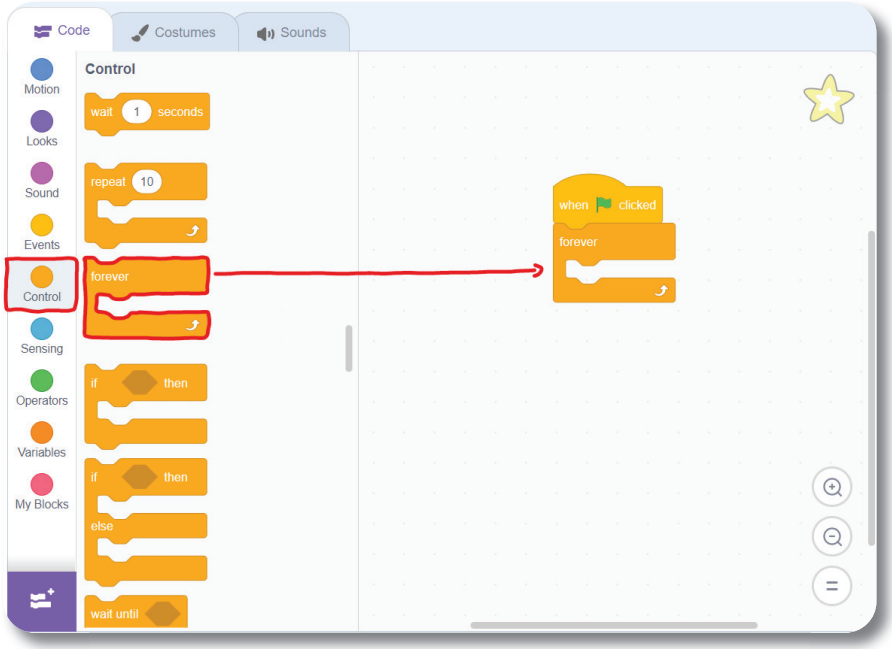
When the game starts, we will get into a “forever” loop. In this loop, we will wait a certain amount of time and drop a byte ball. So for example, if that “certain amount” was 1 second, every second we would drop a byte ball, and this would go on forever.

Another thing we have to do is put a time limit. So after let’s say 30 seconds, the game will stop.

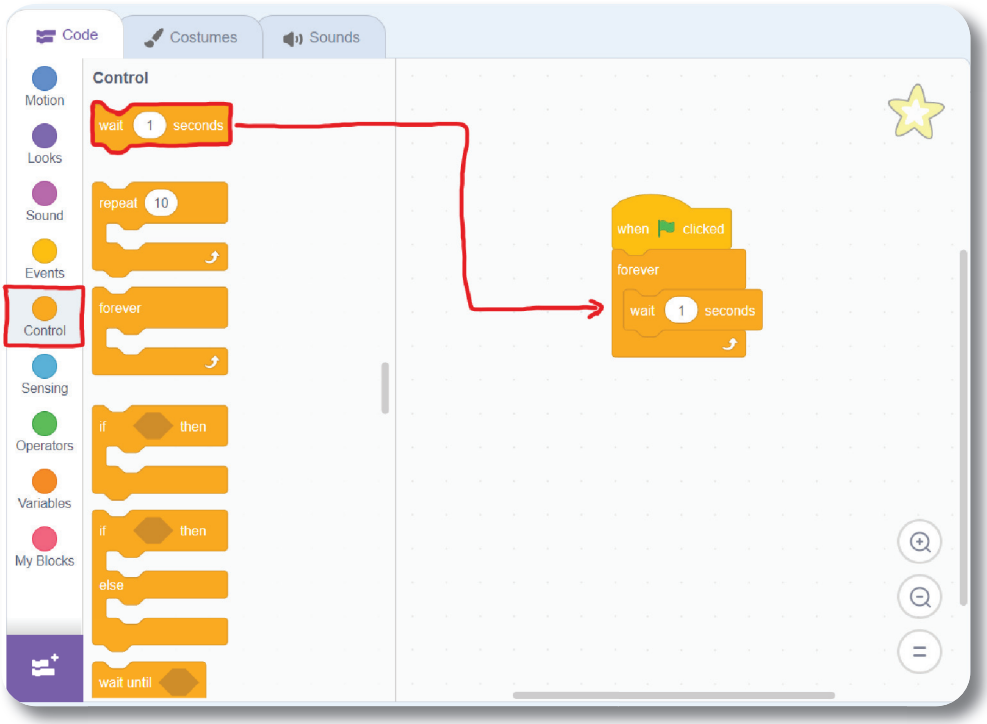
So first, we need to detect when our game starts. To do that, just drop a “when green flag is clicked icon” on your code space.



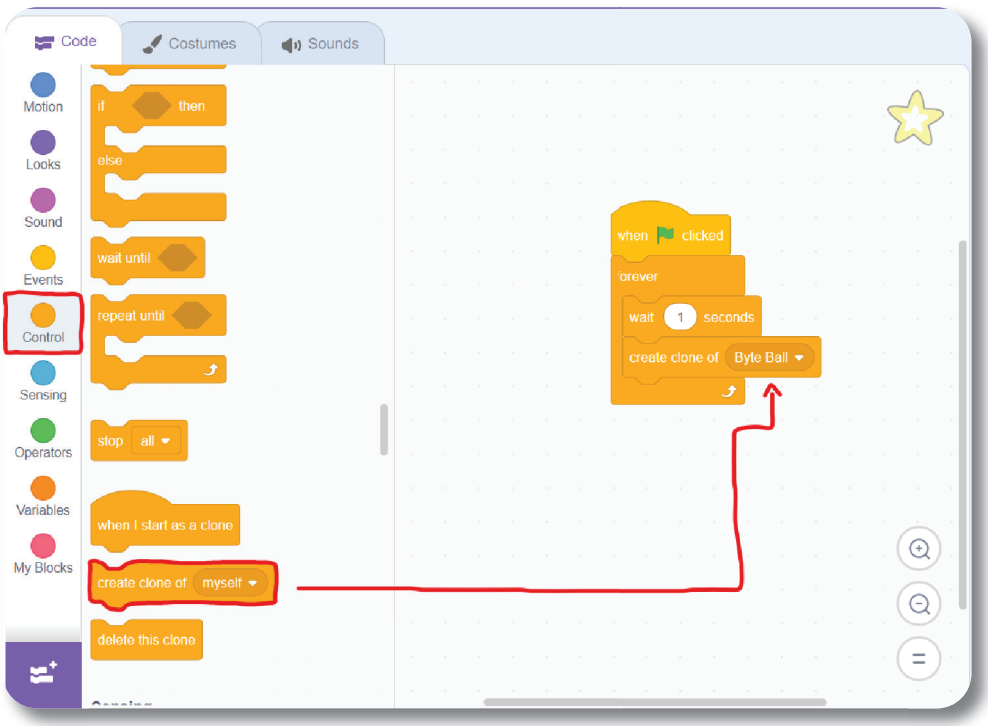
Then, put a “forever” loop under that block. That will start our loop



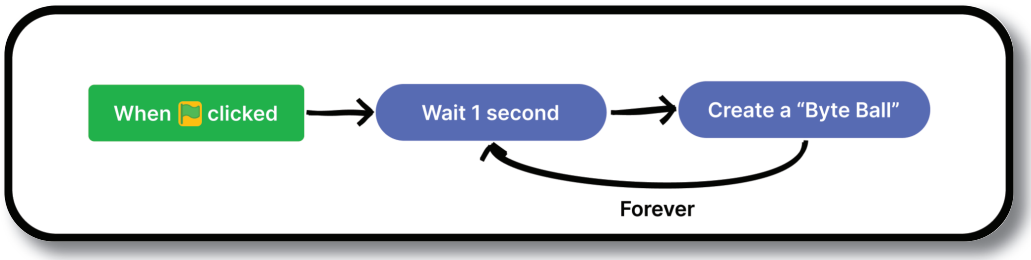
Now put a “wait [value] seconds” block in the loop. This will make it so that the code waits a certain amount of time before moving forward.



Now, last but not least put a “create clone of [sprite]” under the wait block and set the clone to be “Byte Ball”.

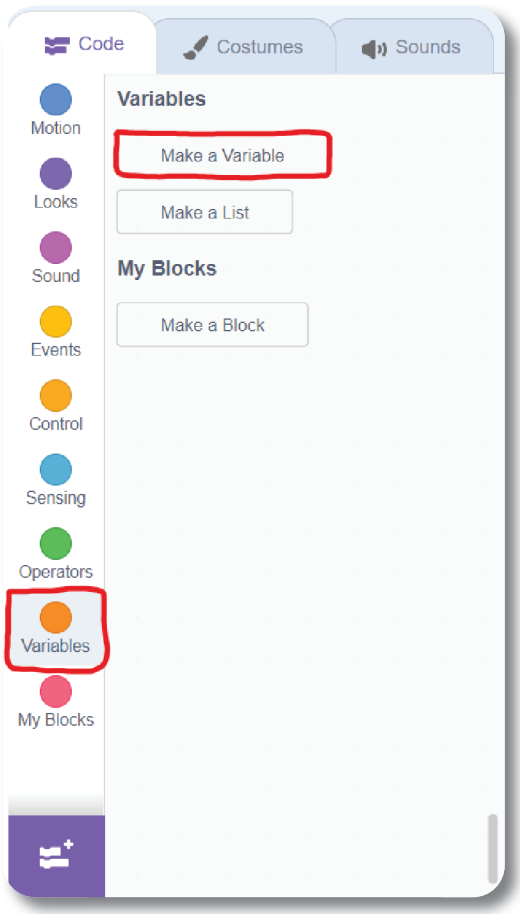


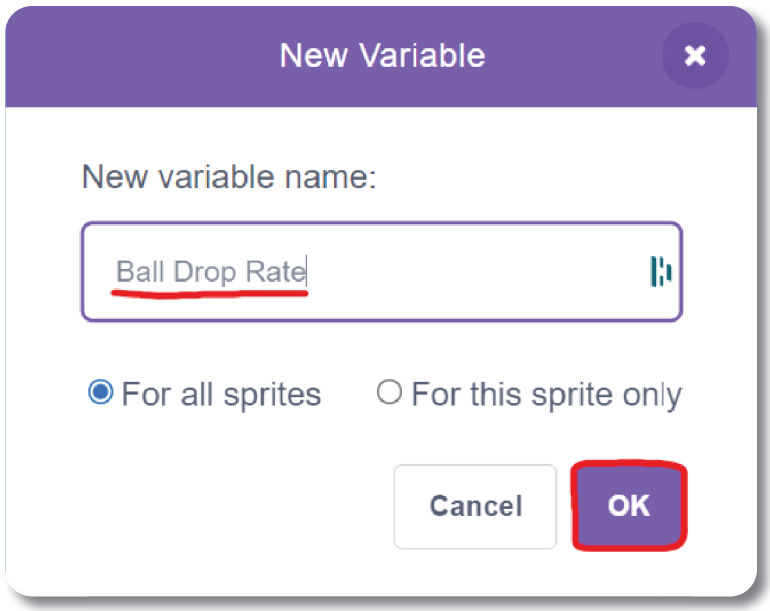
This is what our algorithm looks like now:



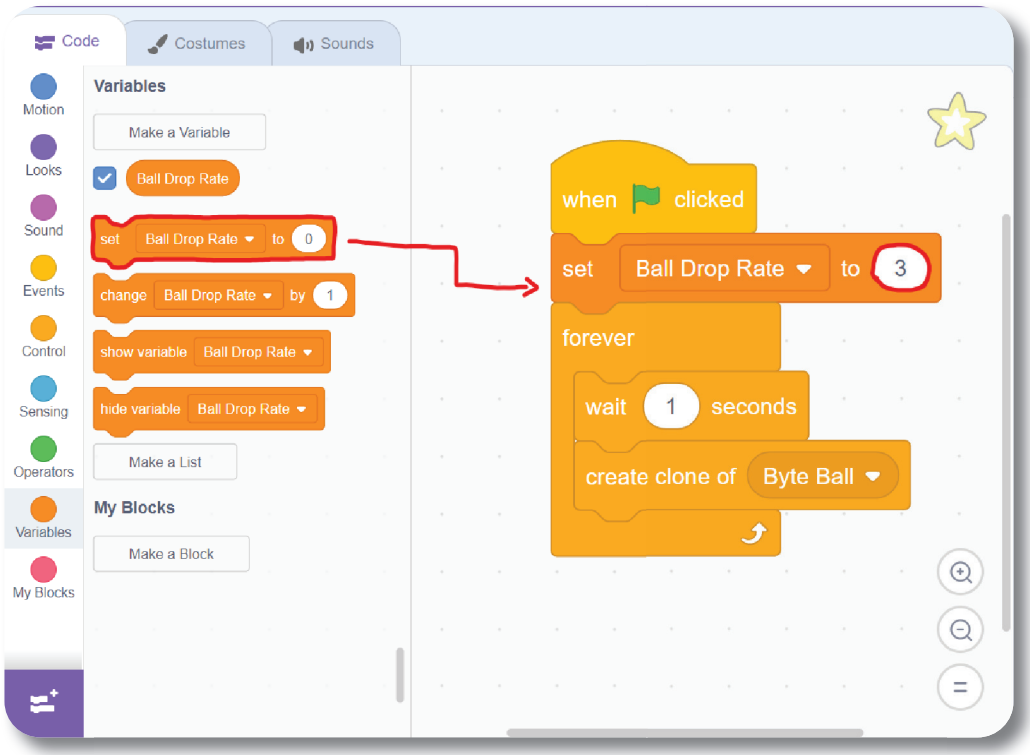
Well, remember our trusty friend "variable"? We will make great use of it in this project. Let's create a "Ball Drop Rate" variable so that we can change it wherever we want, whenever we want.

Go over to the Variables section and create a variable called "Ball Drop Rate".





Now, just drag a "set [variable] to [value]" block and if it isn't already selected, select "Ball Drop Rate" from the dropdown. Change the 0 to whatever you want. I changed it to 3 because I want a ball to drop every 3 seconds.

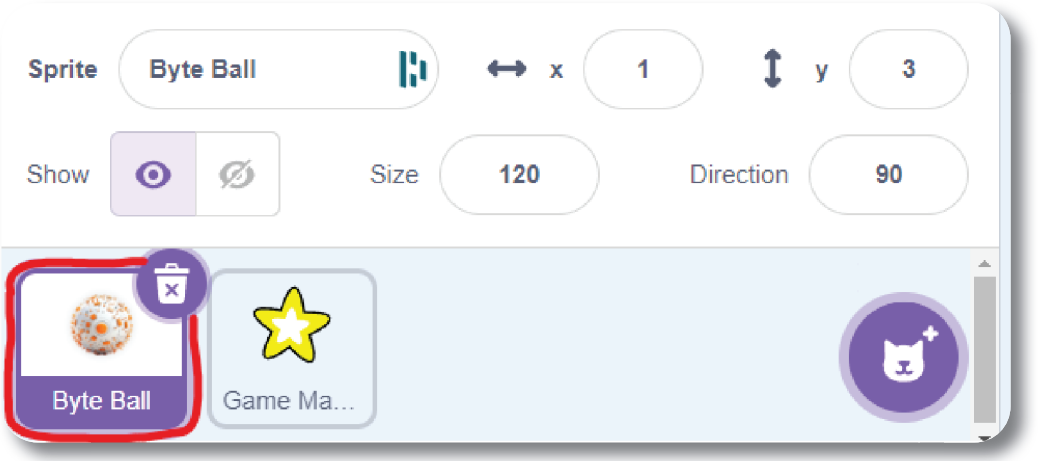


And put the Ball Drop Rate variable as the value in “wait [value] seconds” block.

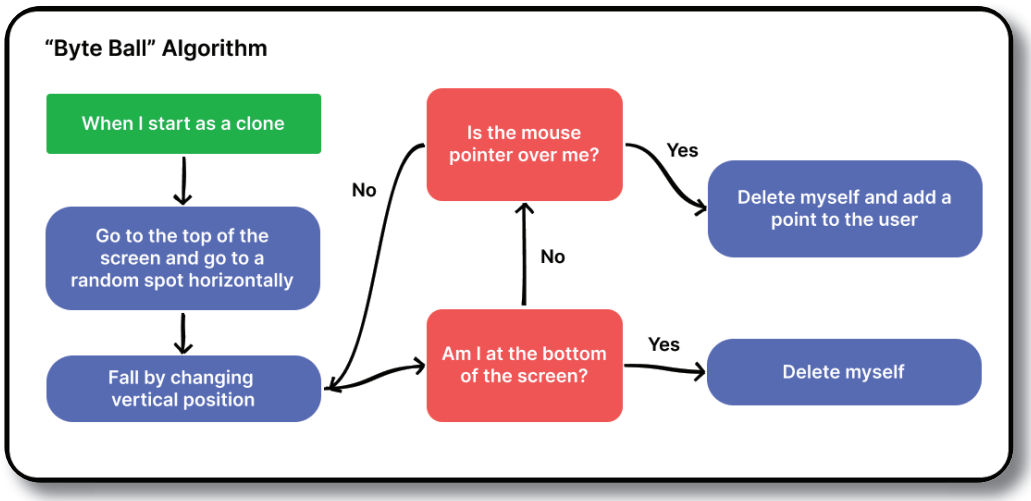
After that, if you start your game, not a lot will happen. You will just see a bunch of balls being created on top of one another. Let’s make them fall!

The Ball Script

It’s time to work on our ball script, which will make our balls select a random position once they are spawned and fall to the ground. On the sprites panel on the bottom right, select our Byte Ball.



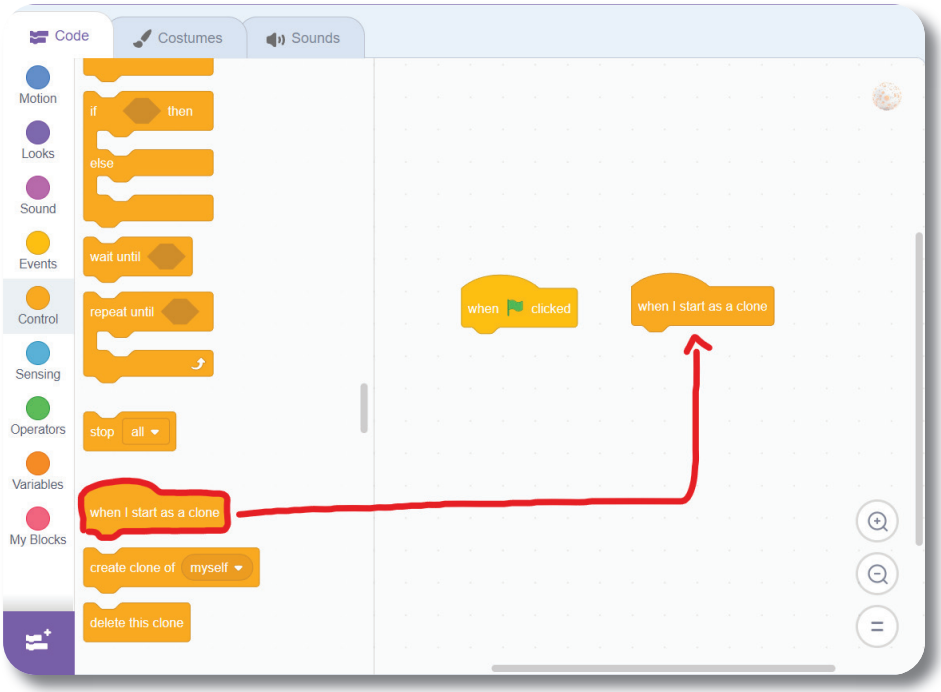
Our algorithm for this ball will be a little bit more complicated. Here is a diagram of what it will look like:



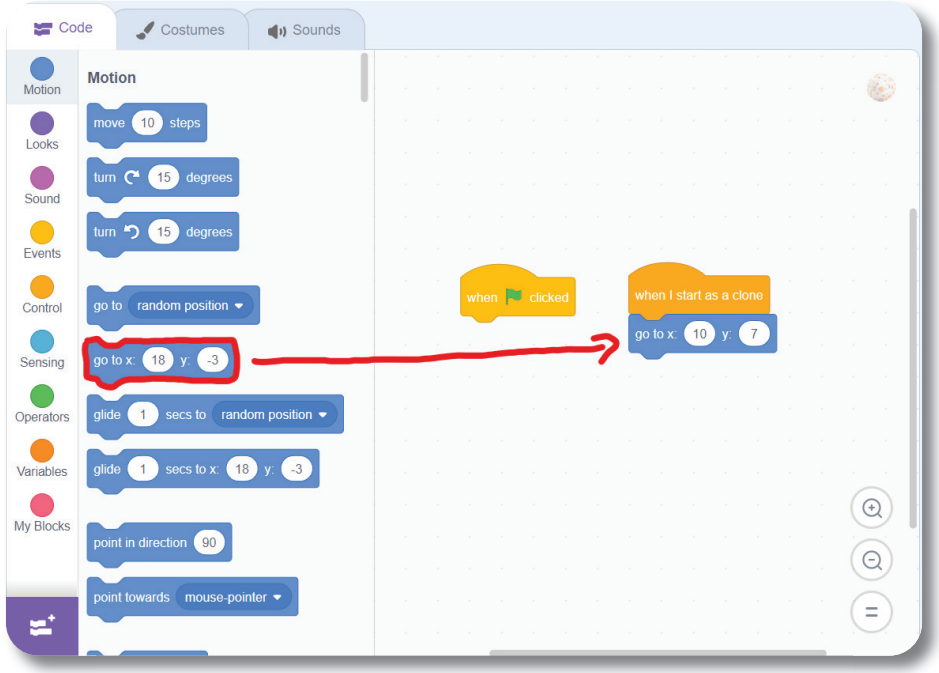
Since our game manager is making "clones" of the Byte Ball, we need to specify to run this algorithm once the Byte Ball starts as a clone. Here is a step-by-step breakdown of the algorithm:

1. Go to a random place on the top of the screen
2. Start falling
3. Am I at the bottom of the page?
    a. Yes: Delete myself without giving a point to the player
    b. No: Move on to Step 4
4. Is the mouse pointer over me?
    a. Yes: Delete myself and give a point to the user
    b. No: Keep falling

So, what are we waiting for? Let's start implementing! First, grab the "when I start as a clone" block from the Control section and drop it in our code space.



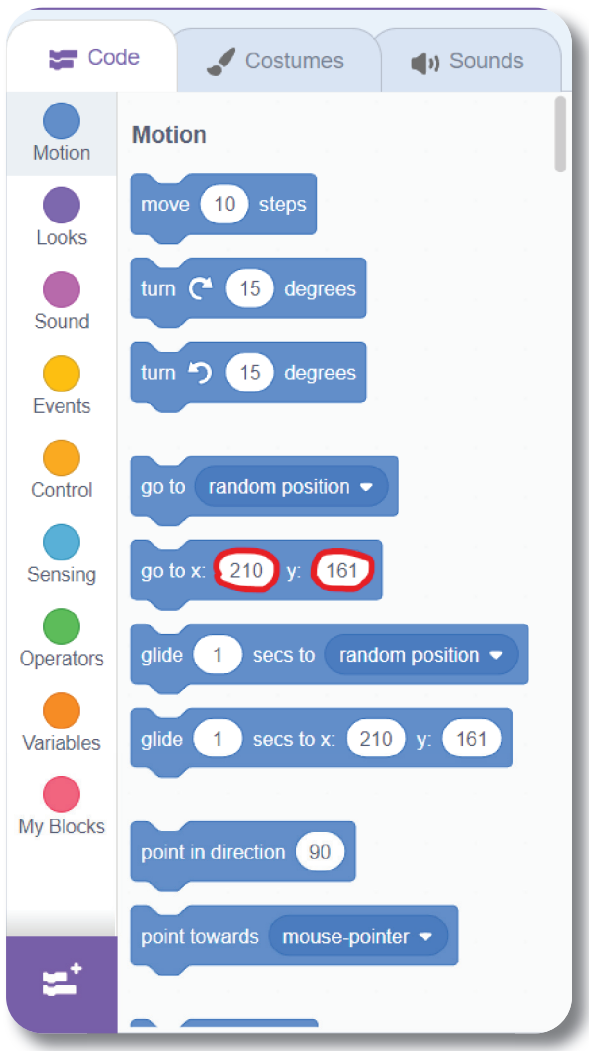
So, when our ball is spawned, it currently sits on top of all of the other balls. Let's make it so that it goes to a random place at the top of the screen. To do that, you need to grab the "go to x:[x] y:[y]" from the Motion section and drop it in our code space.



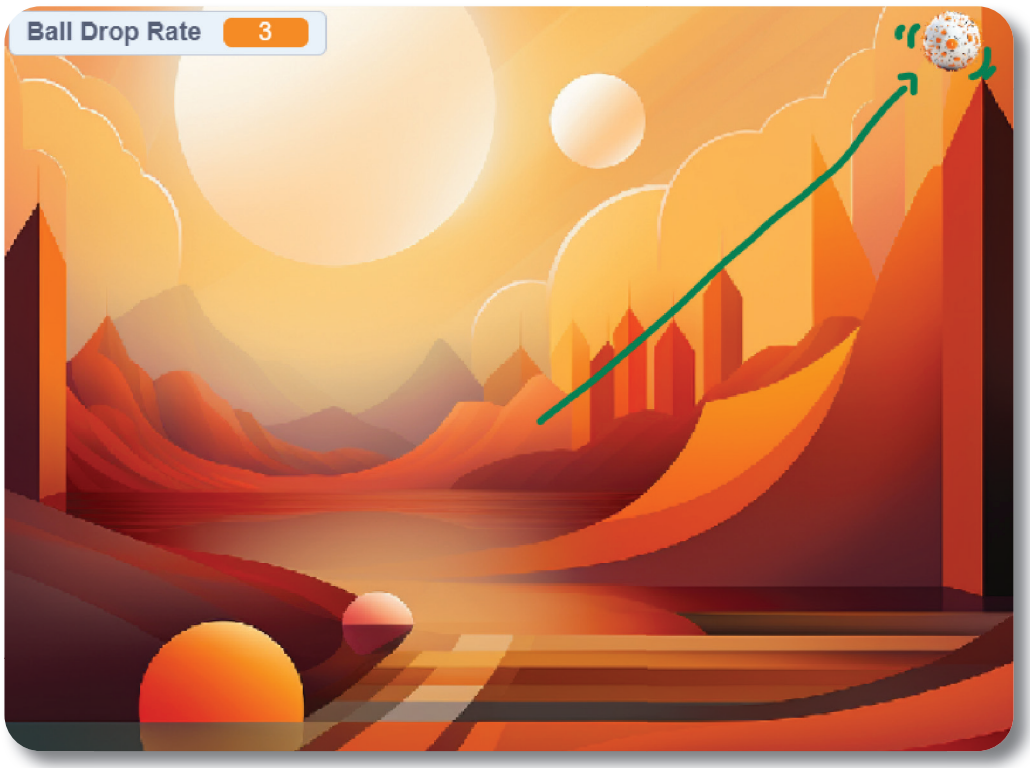
Before moving on, we need to learn what “x” and “y” mean. x, means the horizontal axis. So if our player was running to the left of the screen, we would change the “x” value of the position. “y” means the vertical axis. You can think of this as our height. This is the value we will change when we make our balls fall to the ground. Those values are called “coordinates”.

Here is the first step of the algorithm: Go to a random place on the top of the screen.

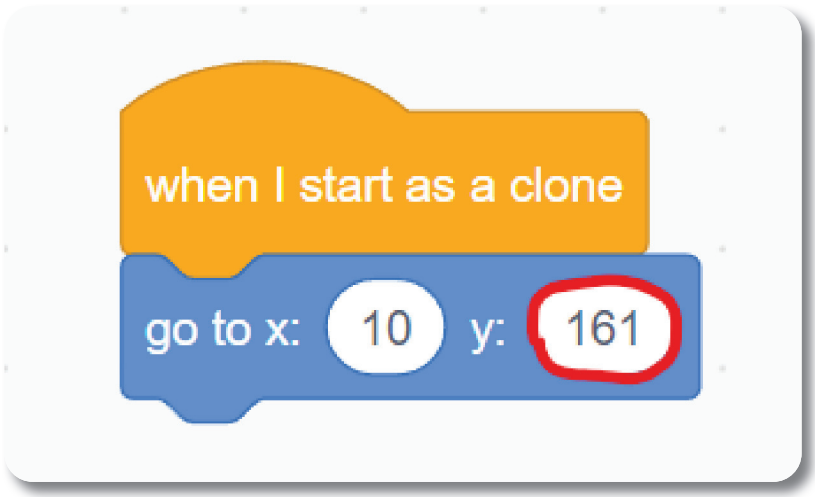
Let’s break it down. First, let’s move it to the top of the screen. Then, we can worry about moving it to a random place.



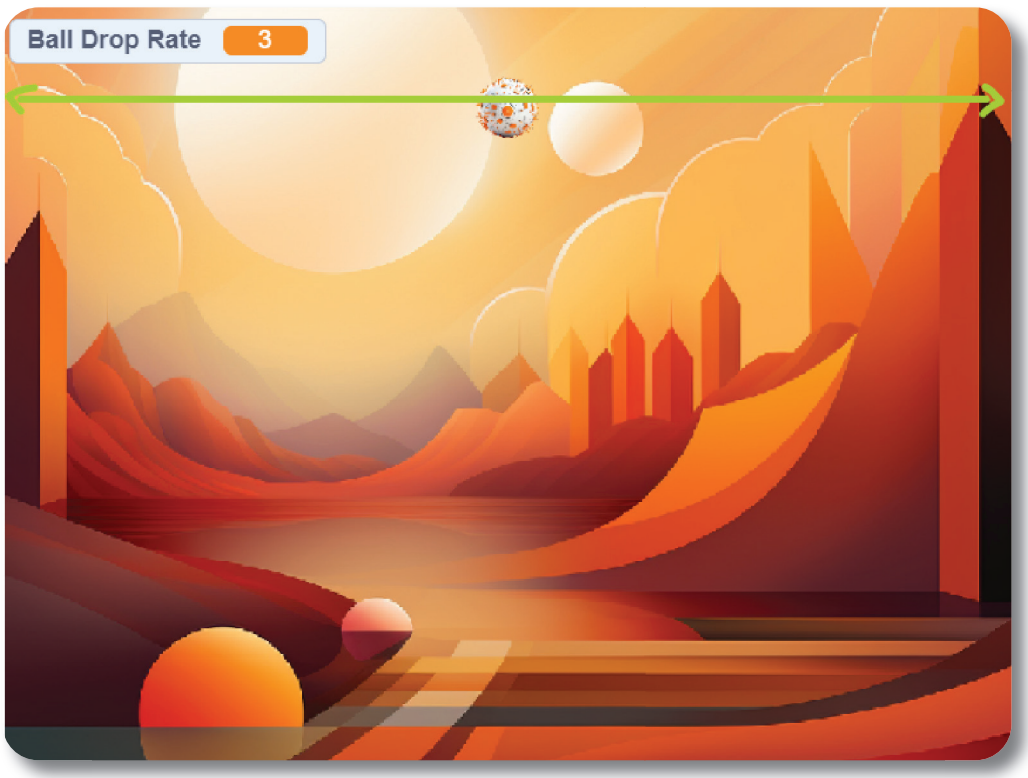
To move it to the top of the screen, we need to know the “y” coordinate of the top of the screen. Move the ball around and you can see that the values “go to x, y” in the code block sidebar start to change. Those values that change represent your ball’s current coordinates. To find the “y” coordinate out, all you have to do is move the ball to where you want it to spawn from and just copy the “y” coordinate.



In the pictures above, I moved the ball to the top and found out the “y” coordinate (vertical position) of the top of the screen is ”161”. You can this or the value you found in the “y” input of the block on the code space.



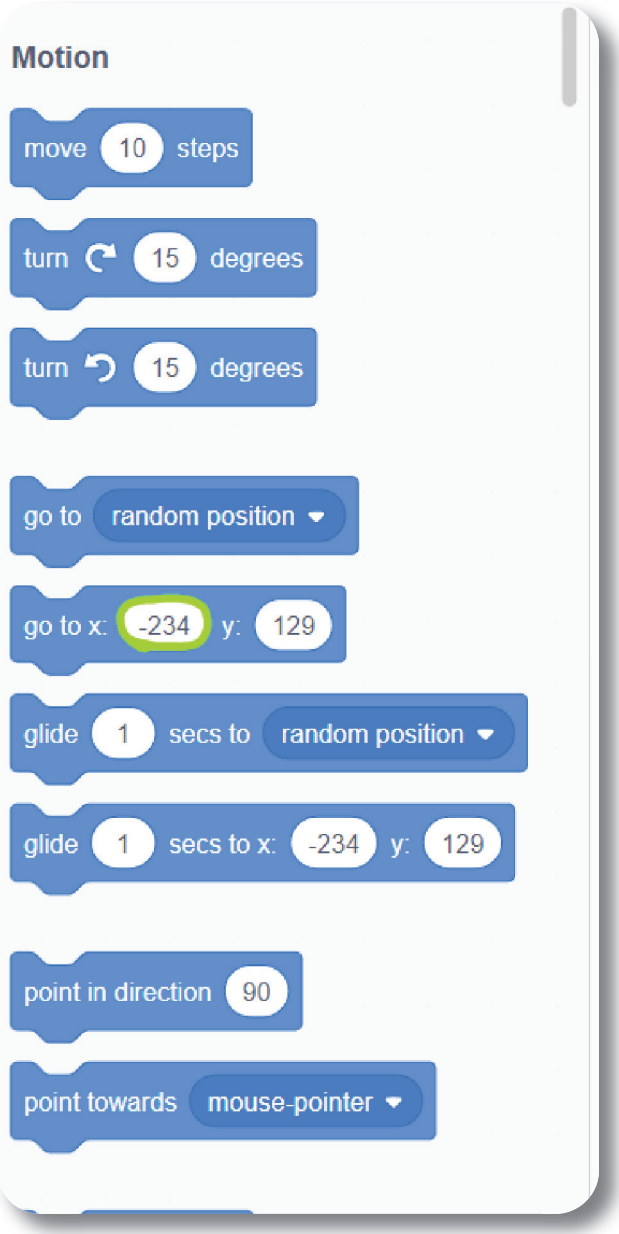
Let's revisit our goal: Go to a random place on the top of the screen. Now that we are on the top of the screen, let's move on to the next part of the goal. For this part of the goal, I want my ball to randomly move to an "x" position (horizontal position) so that my balls don't fall from the same place each time.

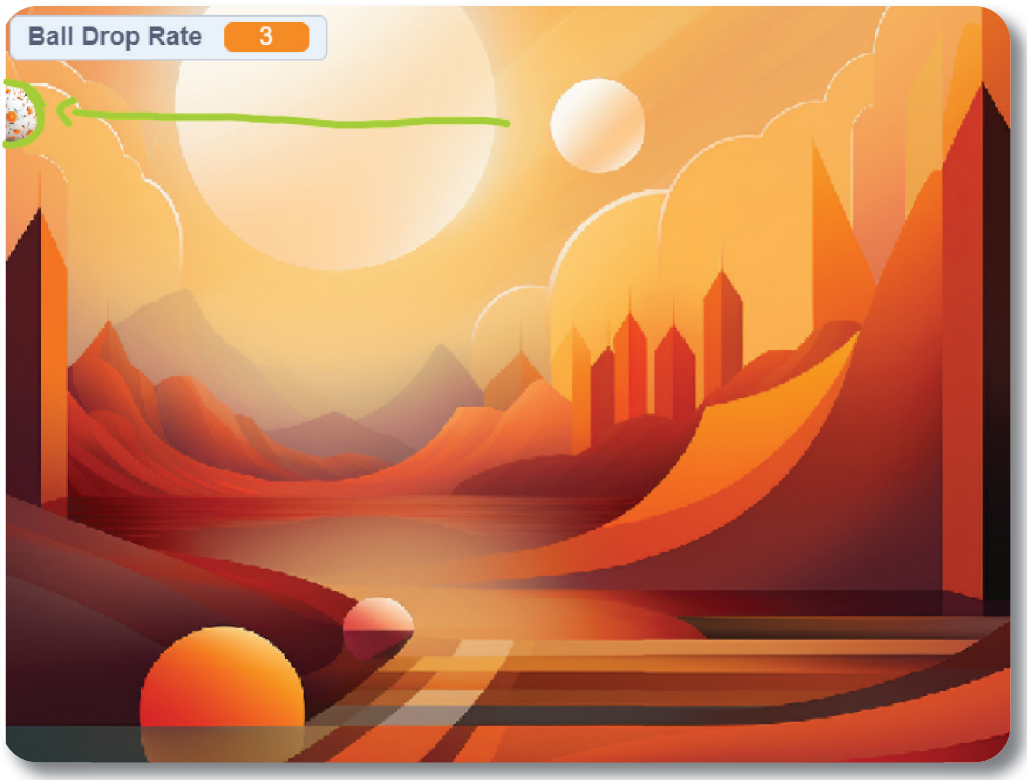


To do that, I need to find out the boundaries of the screen. If we just select a random "x" value without boundaries, our balls might spawn off-screen. To find the boundaries, I just moved the ball to the far-ends of the screen.

The far right:

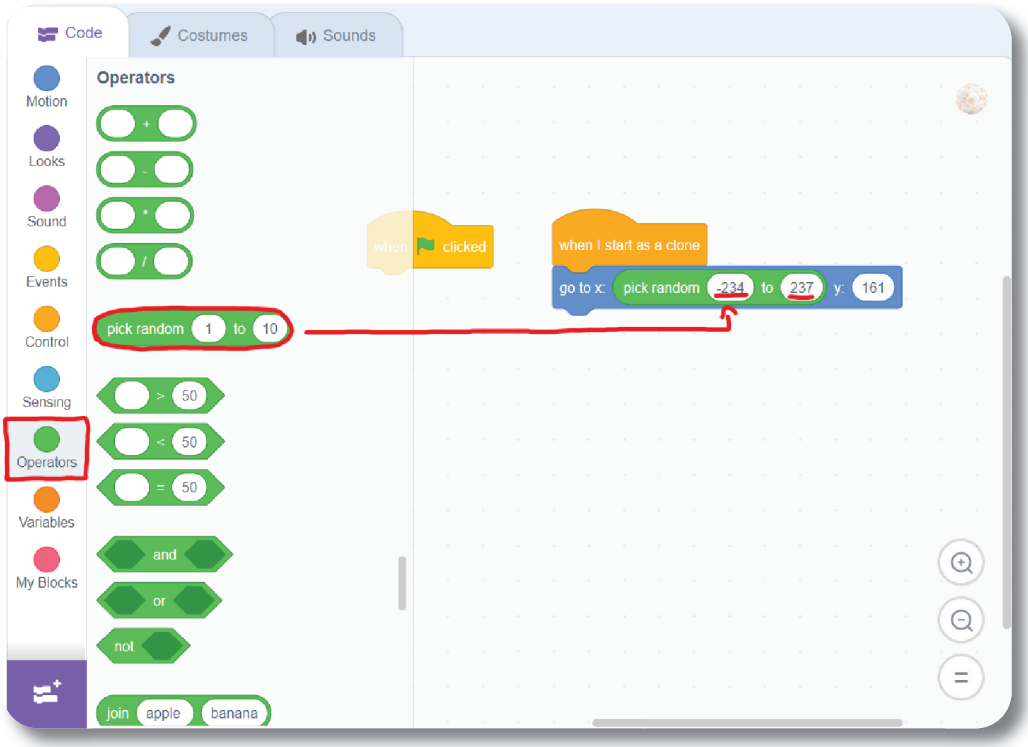
The far left:


Motion
move 10 steps
turn 15 degrees
turn 15 degrees
go to random position
go to x: -234 y: 129
glide 1 secs to random position
glide 1 secs to x: -234 y: 129
point in direction 90
point towards mouse-pointer



These values might be different for you, so I encourage you to try to find the values out yourself.

So, from this whole little experiment, we found out that our screen's boundaries are -234 and 237. Now, all we need to do is pick a random value between these values. Luckily, Scratch has just the block for us! The "pick random [value] to [value]" block that is under the "Operators" section. Grab that block, enter the values above, and put it in the "x" input of our [go to block img].



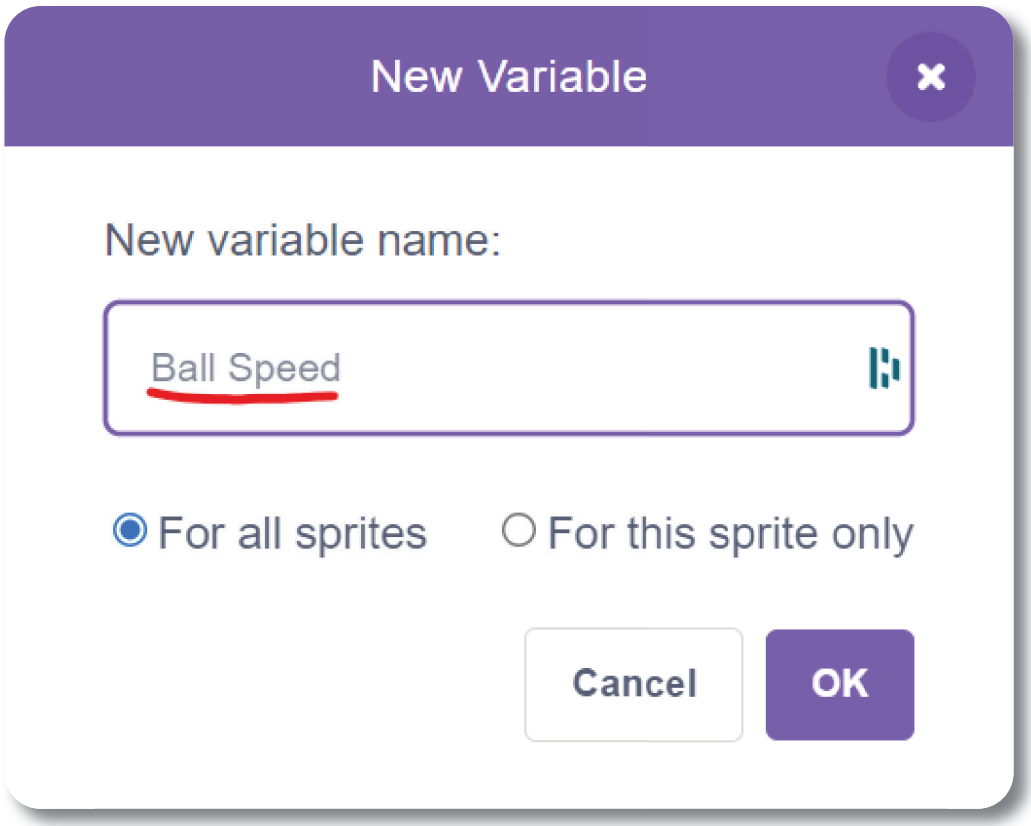
Now, this code is going to move our ball up to the top of the screen and pick a random spot on the horizontal axis when it spawns. We have done the first step of our algorithm! Nice job!

To test it, you need to click on our “Game Manager” and press the green flag (since that is the code that actually spawns stuff).

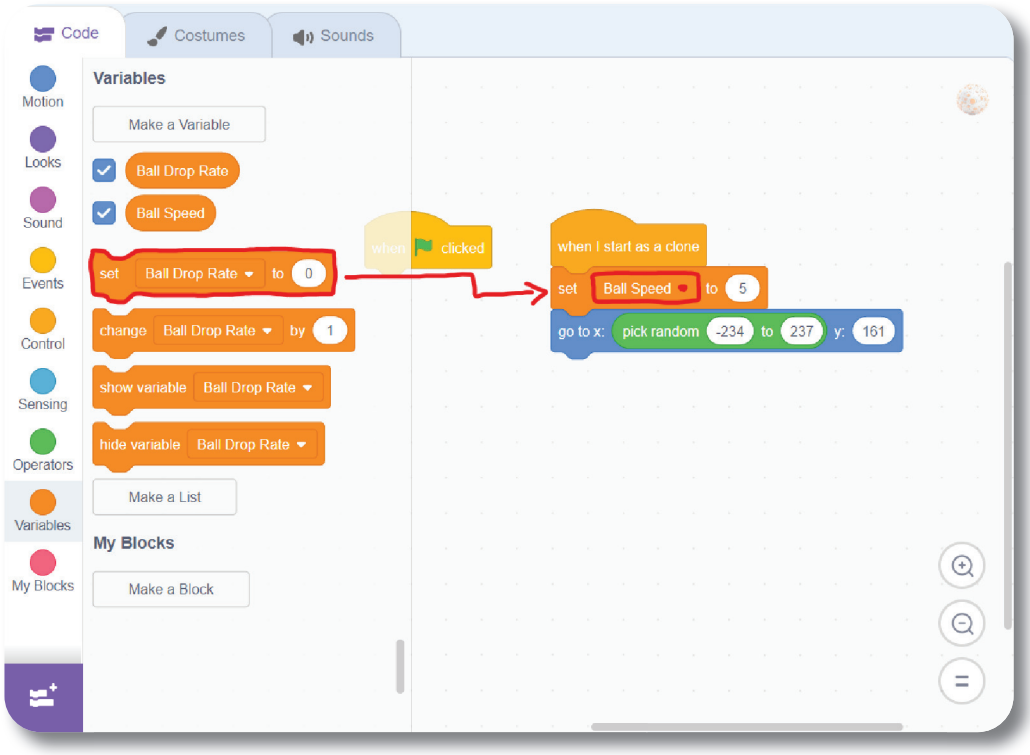
After a few seconds, you should have something that looks like this:

A bunch of random Byte Balls spawning on the sky!

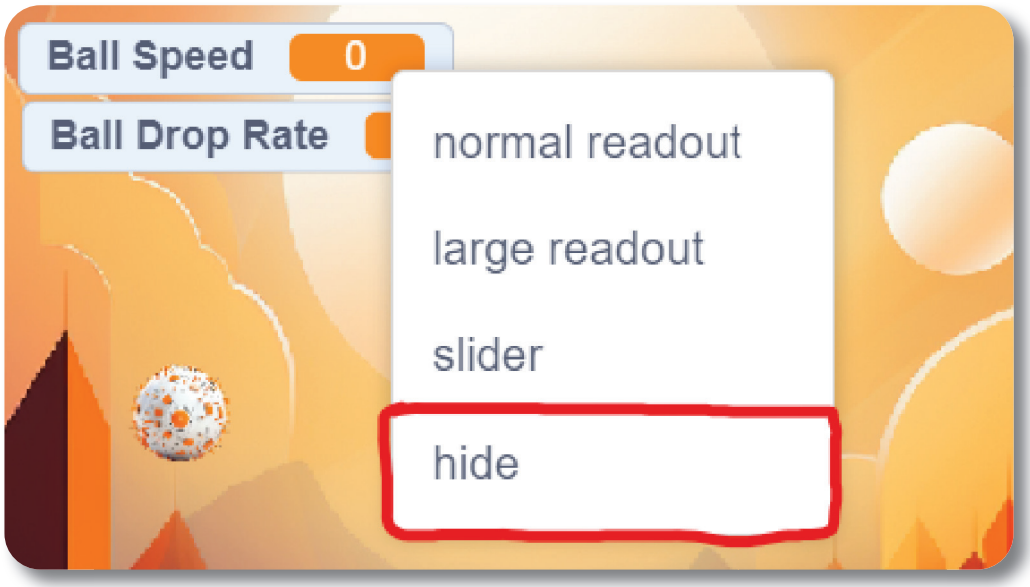
Now, let's make them fall. Before starting out, to make changing the falling speed easier click back on the "Byte Ball" sprite and create a variable called "Ball Speed".



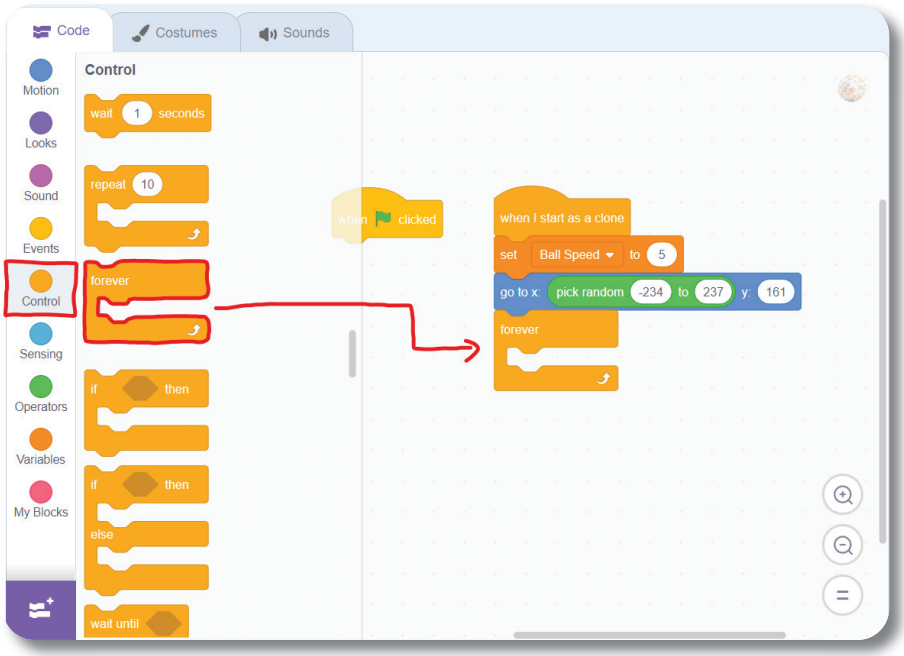
Grab a “set [variable] to [value]” block, select “Ball Speed” from the dropdown, select whatever value you want (I picked 5), and put it at the top of the “when I start as a clone” algorithm.



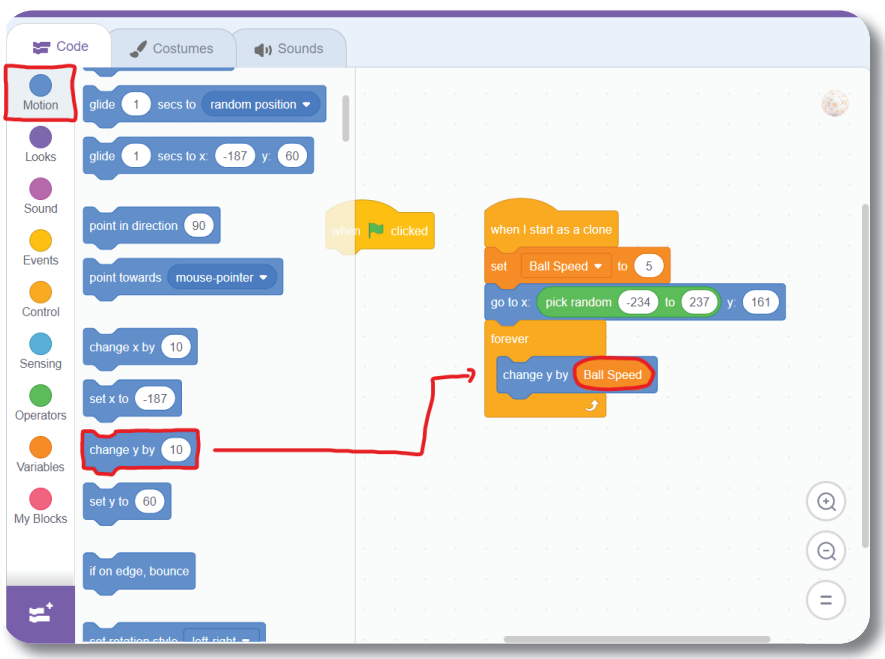
To hide the variable displays showing up, just right click on them and press hide.



Next, put a forever loop under the algorithm.



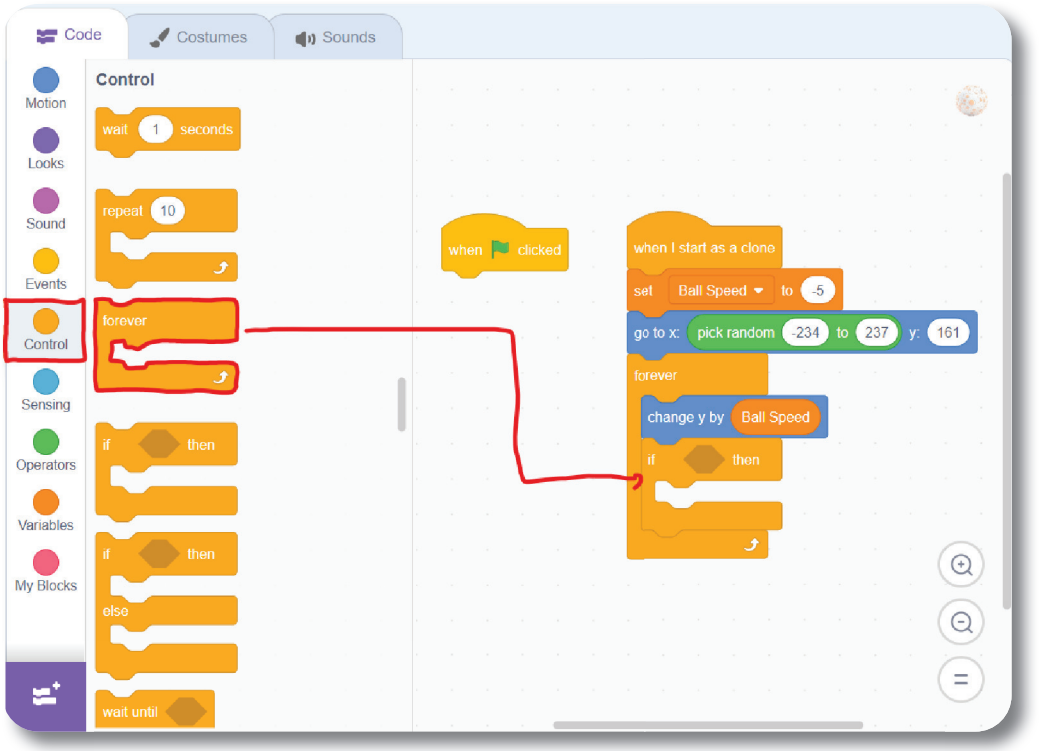
Here, we will continously make our ball fall. To do that, we need to change the height of the ball. What that means is, you guessed it, the “y” coordinate of the ball. To do that, go the the “Motion” section and add a “change y by [value]” block in the forever loop. Then, put your “Ball Speed” variable in the value input.



When you press the green flag, you will notice a problem. All of our Byte Balls are sticking to the top of the screen instead of dropping! Why might that be? Try to figure it out on your own before reading the next paragraph.

Since our "Ball Speed" variable is 5, our code is adding 5 to "y". This causes our balls' height to increase. To fix this, instead of setting our "Ball Speed" variable to 5, we can set it to -5. This will subtract the height instead of adding it, and make out byte balls fall. Let's try it out!

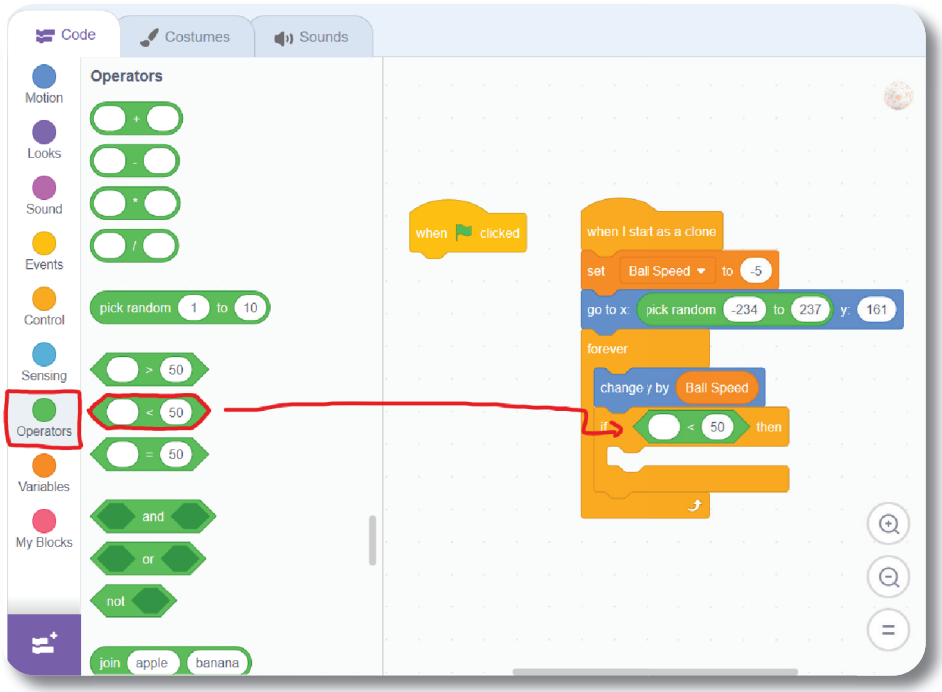
Nice! Our Byte Balls are consistently falling. The next thing we need to do is use our conditional skills to detect either if the ball is on the ground, or if the mouse is touching the ball. Let's start with the ground check first. Start by putting an "if-then" block under the falling logic.



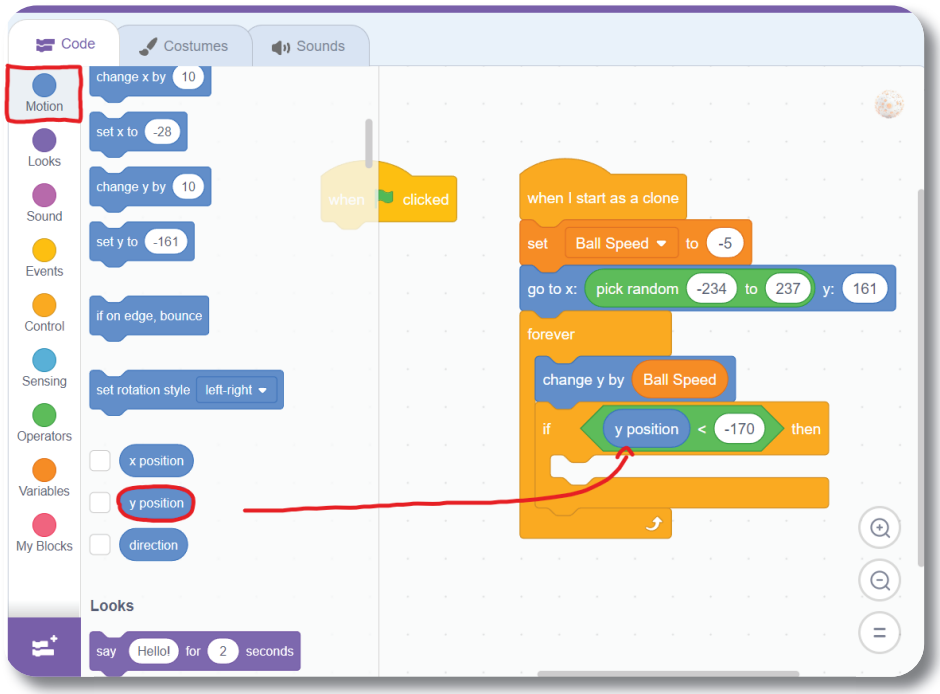
Now, we need to know the "y" coordinate of the ground. You can find it out by using the method that we used to find out the top of the screen or the boundaries. When I used that method, I found the value -170.

What we want in plain English is: "If the y value is smaller than -170, then delete yourself."

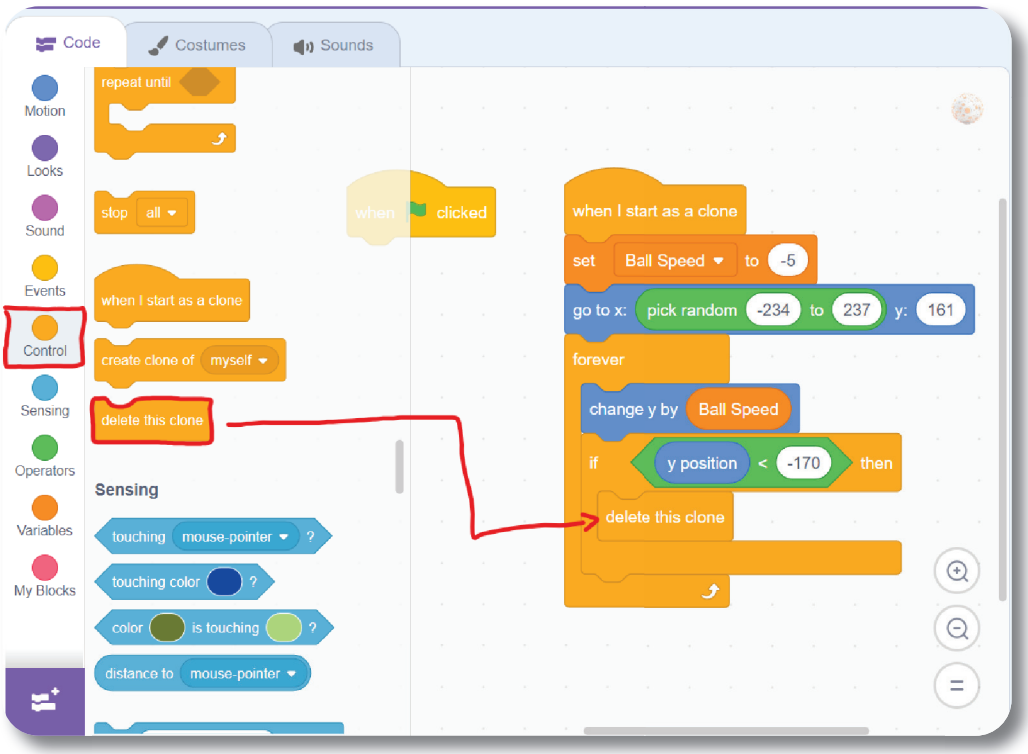
To do this, we need to compare our current y value to -170. To do the comparison, grab a "[value] < [value]" block from the "Operators" section and put it in the space between the "if" and the "then".



Then, type -170 (or whatever value you found with your trial and error) in the place of where 50 is. Then, put the "y position" variable on the left side of the comparison.

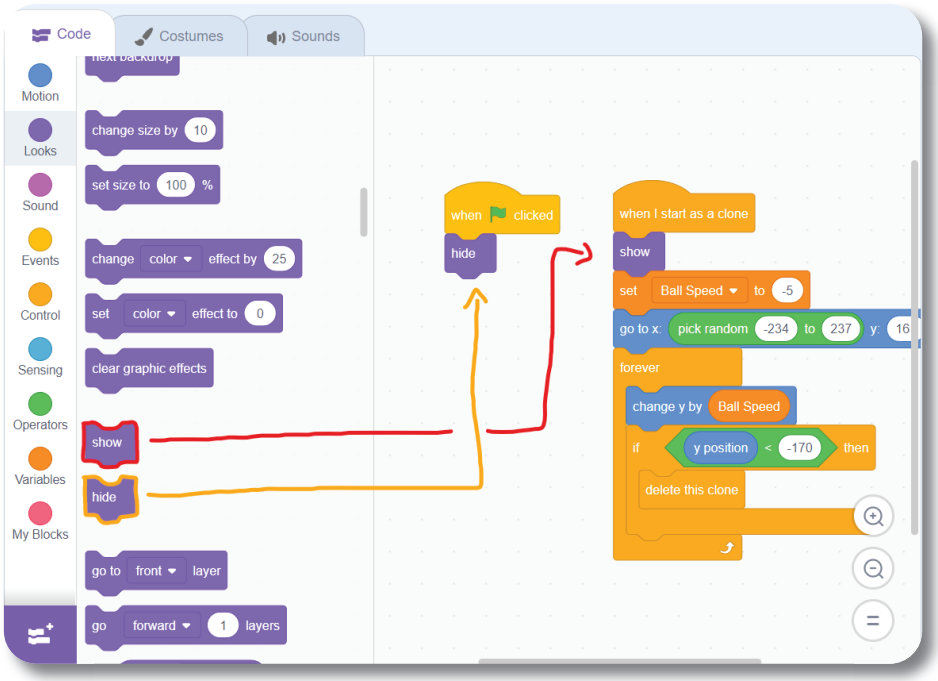


Next, put in the “delete this clone” block that is under the “Control” section under that if statement.

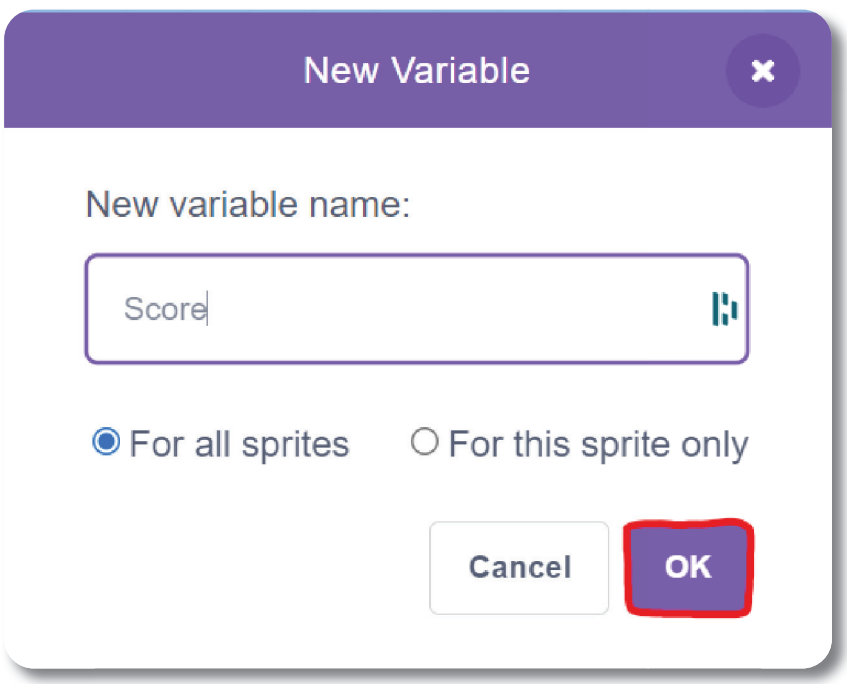


After that, press the green flag to test your code. You should see Byte Balls disappearing once they hit the ground. Great job!

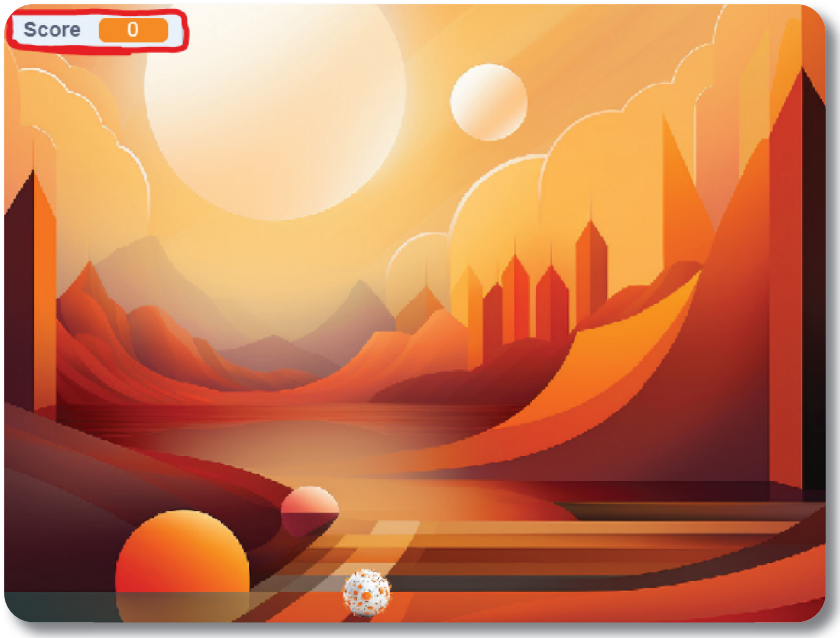
You might have realized that a ball stays at the same place during the game. Before moving on with the script, we need to hide the ball that is already on the screen. So when the green flag is clicked, we will hide the ball and when another ball starts as a clone, we will show that ball.



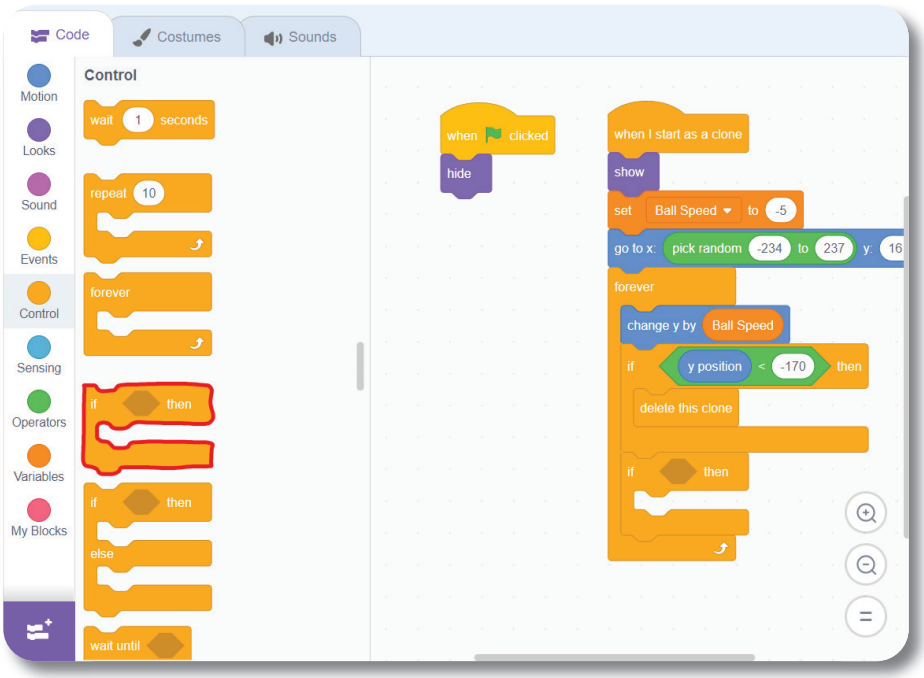
The last thing we need to do on this Byte Ball script is to detect when our mouse is on the ball so we can delete the ball and add to the score of our player. First, create a "Score" variable.



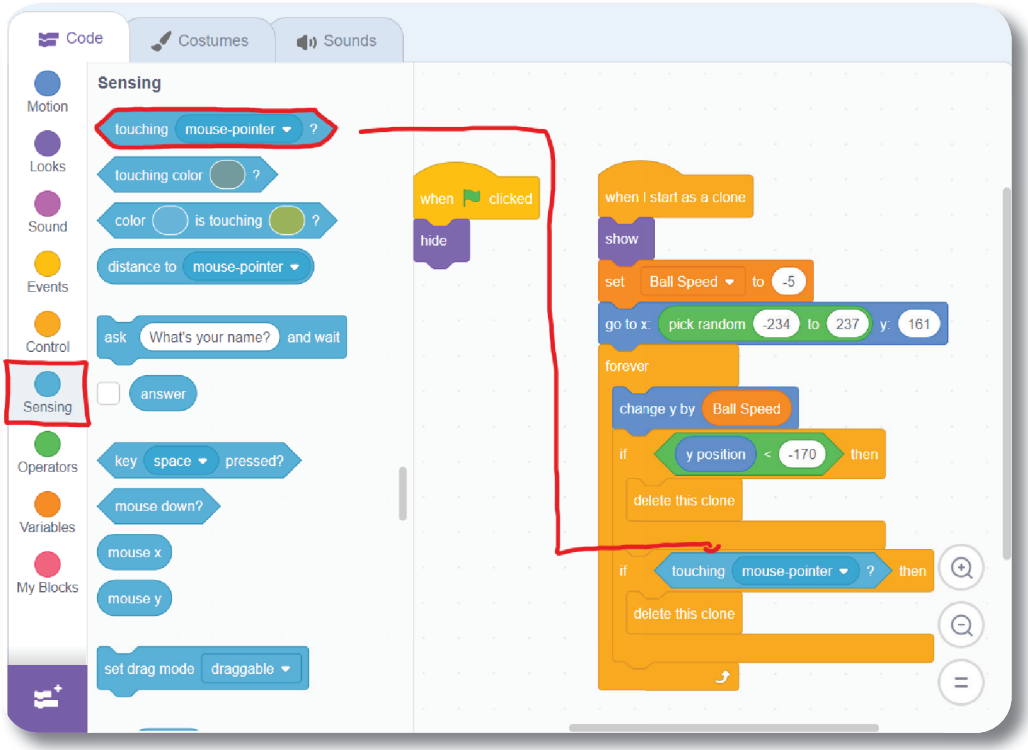
Now, unlike the other variables, we want this to show up on the game screen so that the player knows what their score is. So if it's hidden by default for some reason, make sure to unhide it. It should look like this:



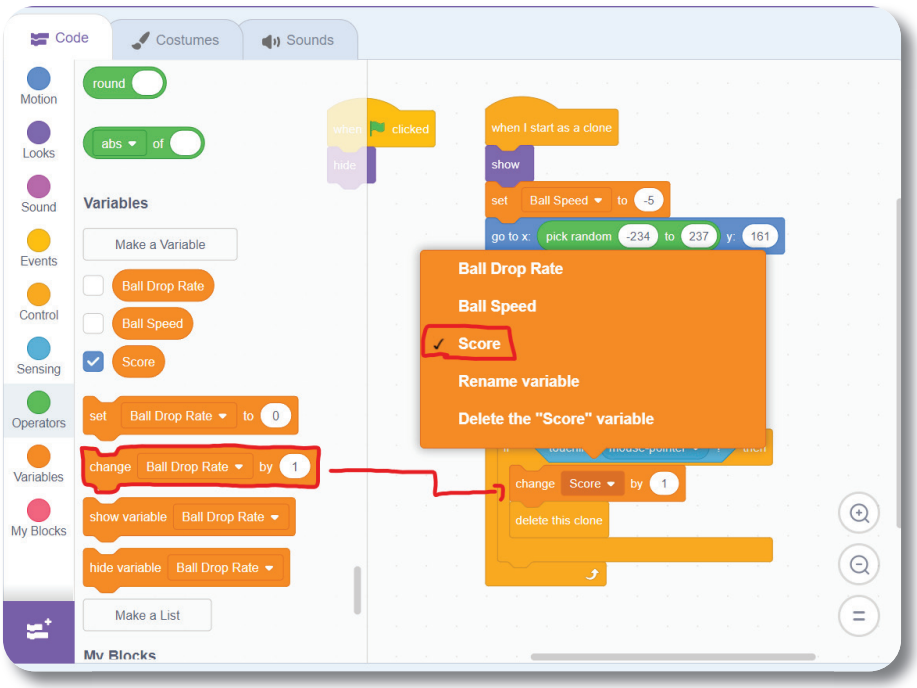
Next, put another if-then block under the ground-detection-if-block (yes, that is what I named it).



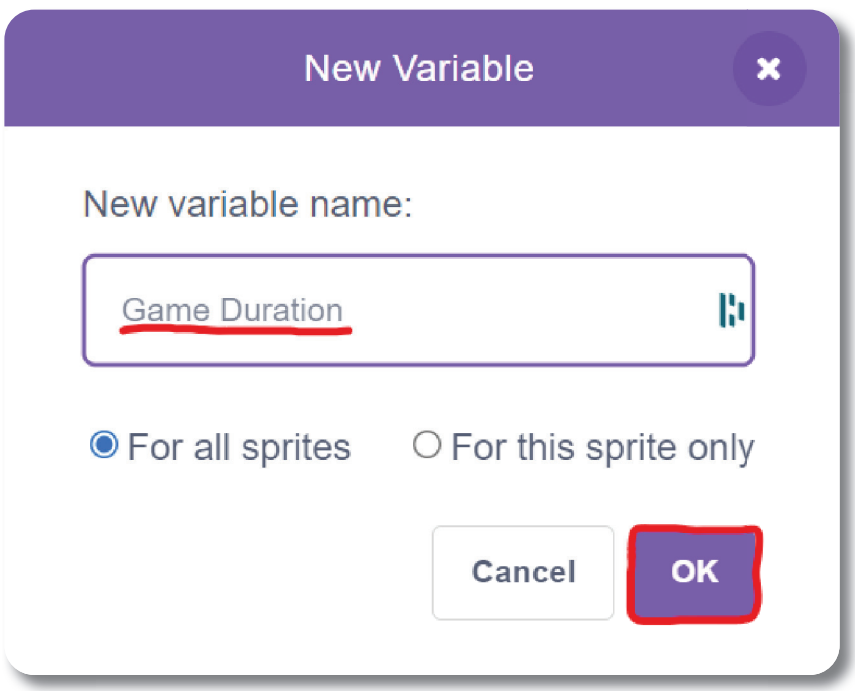
Now, go over to the “Sensing” section. There, you will find the “touching mousepointer?” block. Put that in between the “if” and the “then”. Also, put the “delete this clone” in this block as well like we did with the above block.



The difference between the above if block and this if block will be that in this if block, we will add score to our user. To do that, go over to the “Variables” section and get a “change [variable] by [value] block”. To change the “Score” variable, select score from the [variable] dropdown menu.

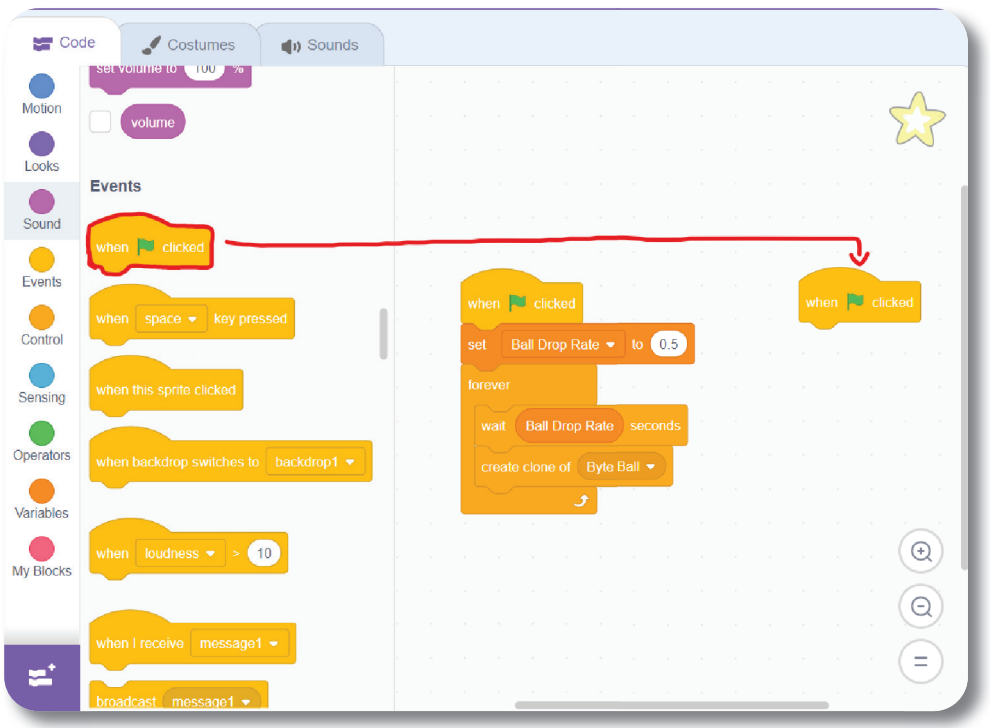


Finally, we will add a time limit to our game. Create a variable named “Game Duration”.

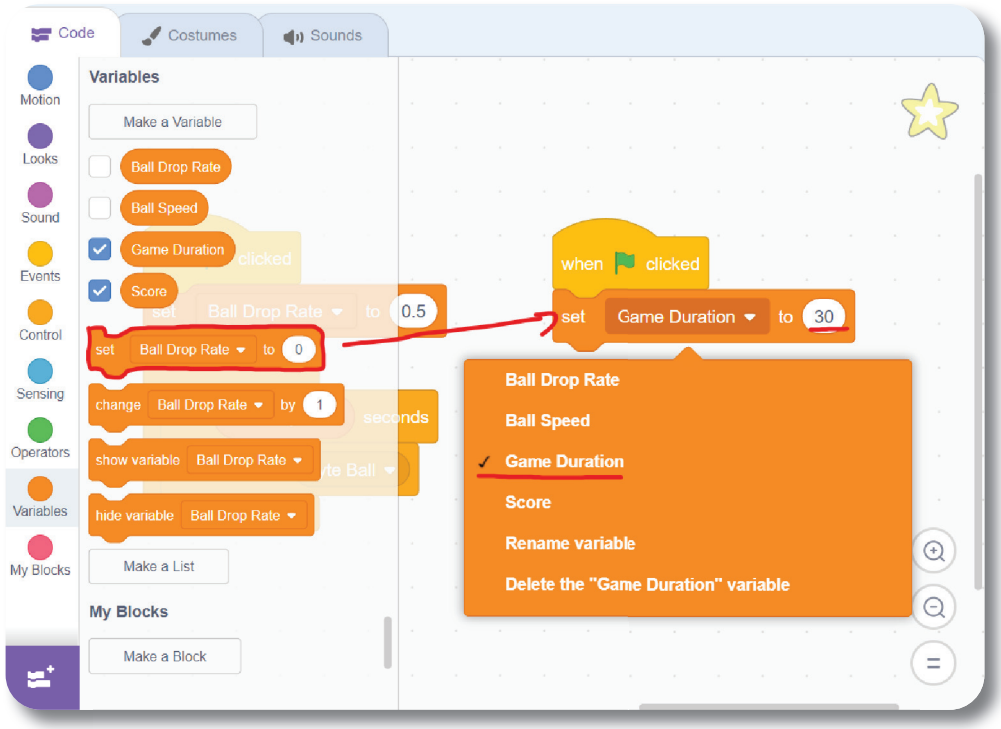


Remember our Game Manager. It was a star in my case. Whatever yours is, click on it to edit its script.

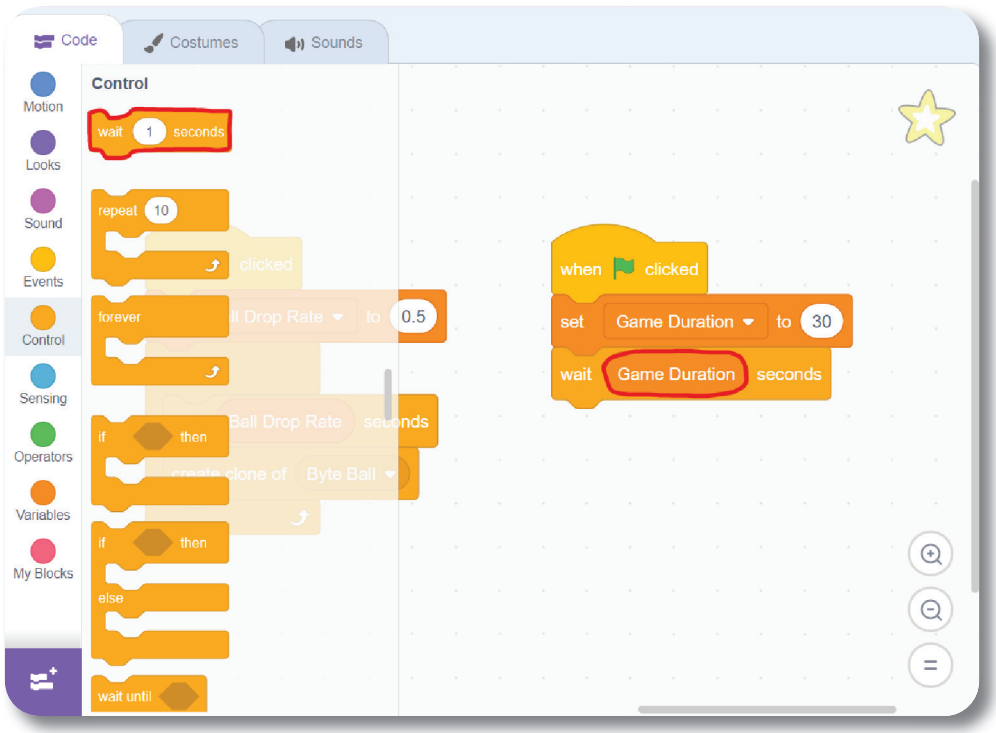
Grab another “when green flag clicked” block, since we can’t add more blocks to our current “when green flag clicked” block due to the forever loop.



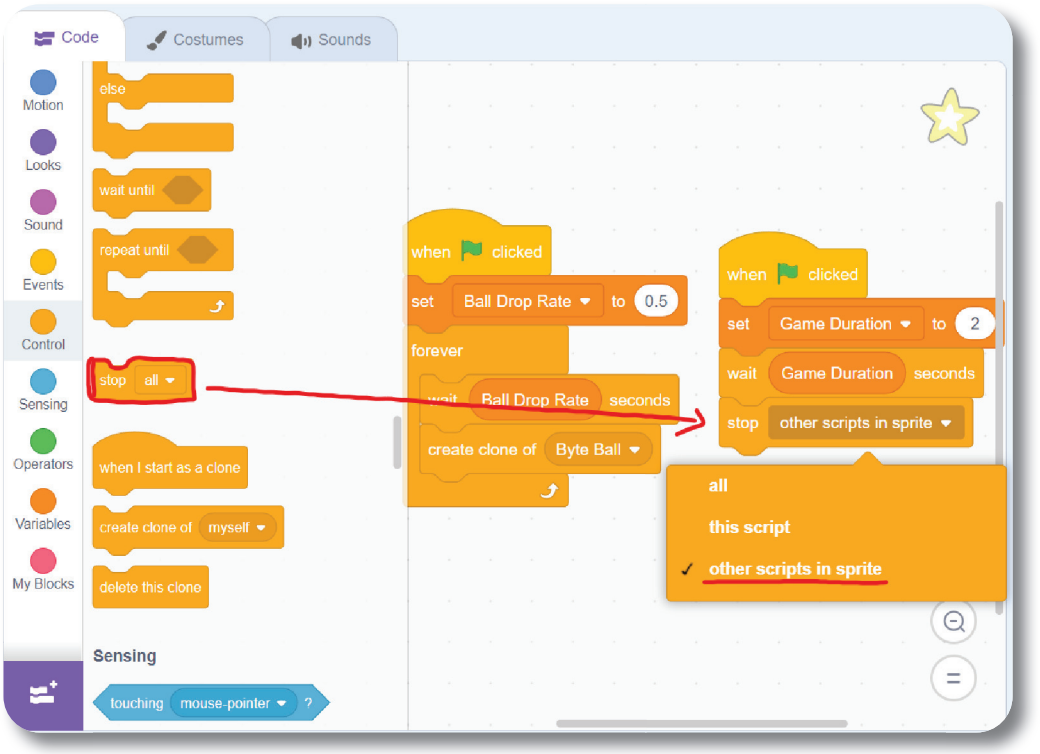
Put a “set [variable] to [value]” block under it. Select your “Game Duration” from the dropdown and set the value to whatever you want. I set it to 30, so my game is going to run for 30 seconds.



Under that, put a “wait for [value] seconds” block and put in your “Game Duration” variable in place of the [value].



After we wait a certain amount of time (30 in this case), we will just stop our game. To do that, go over to the “Control” section and scroll down to the “stop [scripts]” block. This will stop scripts we tell it to stop. We want to stop the other script on this sprite, which is the script that spawns balls. Drag and drop it under the “wait [Game Duration] seconds” block and select the “other scripts in this sprite” option from the dropdown.



And Voilà! We did it! I am so proud of you for fearlessly taking your first steps into the exciting and growing field of coding. Give yourself a pat on the back, and share your game with some friends!

CHAPTER 8:

Career Paths In Coding

Welcome to the final chapter of our coding journey! By now, you've dabbled in the basics of coding, built your own projects, and unleashed your creativity. But this is just the beginning. Coding is not just about solving puzzles or creating games; it's a skill that can open doors to numerous exciting career paths. Let's explore the vibrant world of opportunities that coding skills can lead to.

Software Developer: Turning Ideas into Apps

Software developers are the creative minds behind computer programs and applications. Using coding languages more advanced than Scratch, like Python, Java, or C++, they design and build software that solves real-world problems. Whether it's creating an app to help people stay organized or developing a complex software system for businesses, software developers are at the heart of the digital world.

Web Designer: Crafting the Digital Space

Ever wonder who designs the websites you love visiting? Web designers use coding to create visually appealing and user-friendly websites. They blend artistic skills with technical coding knowledge, ensuring that websites are not only beautiful but also functional and easy to navigate.

Game Developer: Bringing Imaginations to Life

If you loved creating games in Scratch, imagine doing it on a larger scale! Game developers use coding, art, and storytelling to create engaging video game experiences. From mobile games to console blockbusters, game developers turn imaginative ideas into playable realities.

Data Scientist: The Data Detective

Data scientists are like detectives, using coding to sift through massive amounts of data to find patterns, make predictions, and solve complex problems. They play a crucial role in various fields, from healthcare to finance, helping organizations make data-driven decisions.

Robotics Engineer: Merging Coding with the Physical World

Robotics engineers use coding to create robots that can perform various tasks, from manufacturing cars to exploring other planets. If you're fascinated by robots and want to see your code come to life in the physical world, robotics engineering might be your calling.

Cybersecurity Expert: The Digital Guardians

In a world where much of our lives are online, cybersecurity experts are essential. They use their coding skills to protect computer systems from hackers and cyberattacks, ensuring that sensitive information remains secure.

Artificial Intelligence Developer: Shaping the Future

AI developers use coding to create systems that can learn and make decisions, mimicking human intelligence. From self-driving cars to virtual assistants like Siri and Alexa, AI developers are at the forefront of technological innovation.

Freelancer: Your Code, Your Rules.

Love the idea of being your own boss? As a coding freelancer, you can take on projects that interest you, work from anywhere, and set your own schedule. Whether it's building websites, developing apps, or offering consulting services, freelancing offers flexibility and variety.

The Future Awaits!

Remember, the world of coding is constantly evolving, and there are always new things to learn and explore. The skills you've started developing here are just the foundation. Whether you dream of developing the next hit video game, designing cutting-edge websites, or launching a startup, your coding journey can take you there. Keep learning, keep coding, and most importantly, keep dreaming big. The future is yours to code!