CoderDojoKC Documentation

Release 0.0.1 alpha

CoderDojoKC

Oct 12, 2019

CONTENTS:

CHAPTER

ONE

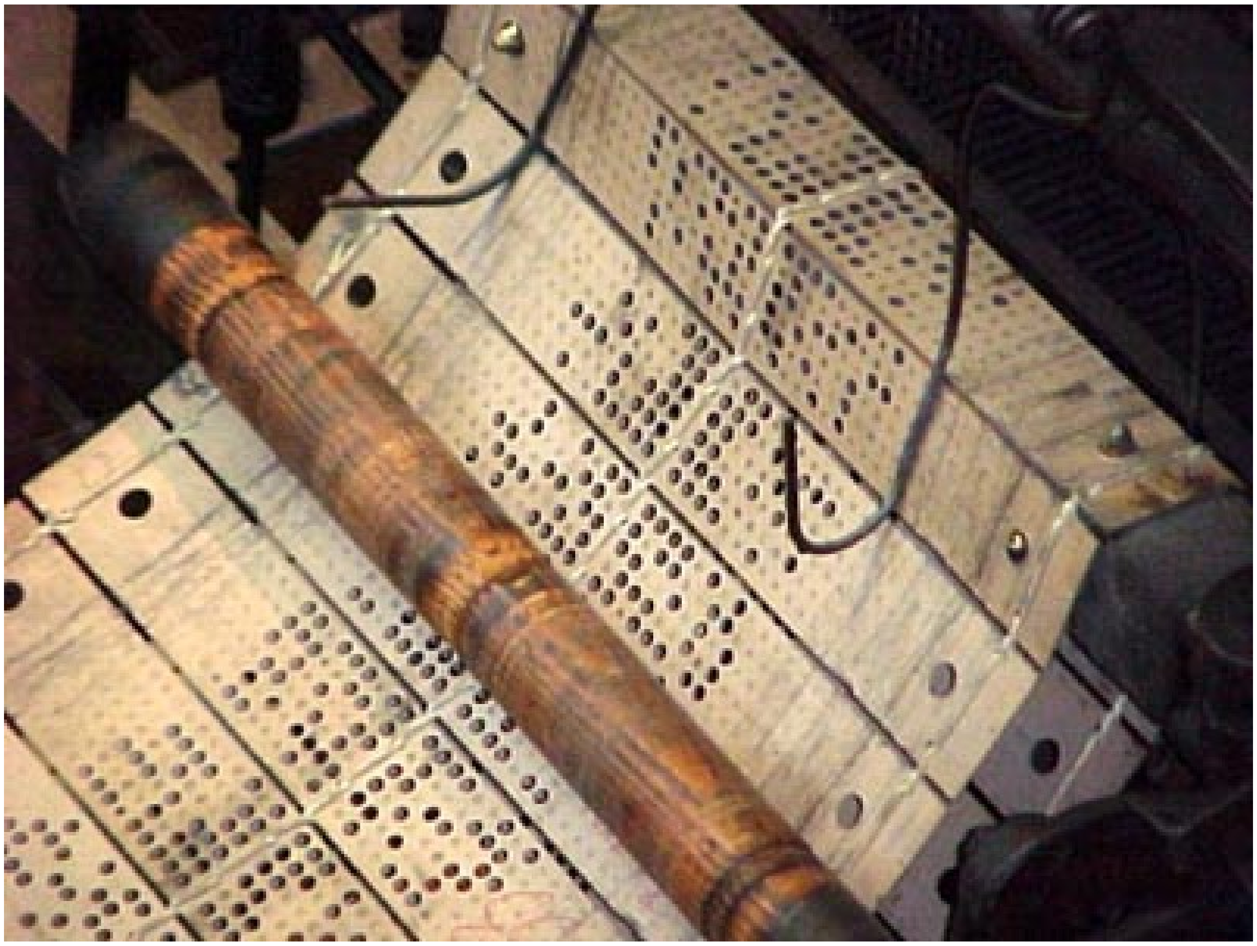
DRAWING WITH CODE

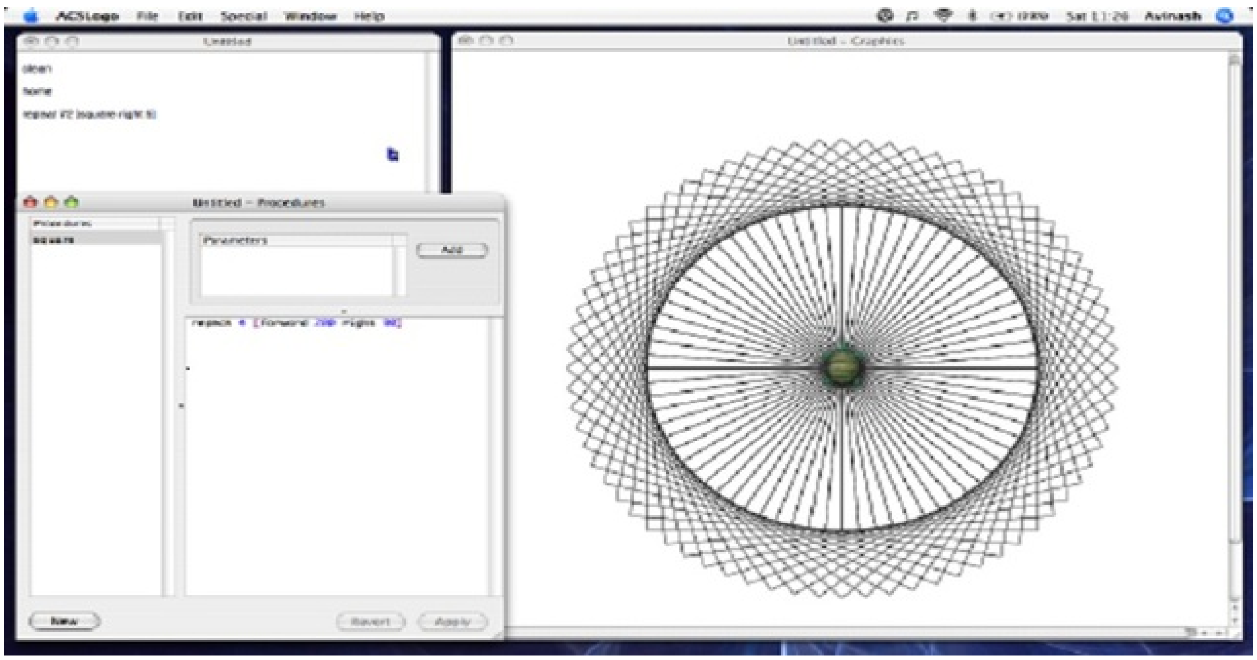
Up through the mid-20th Century, software was written by punching holes in cards in specific patterns and feeding those cards into a computer in a specific order.

Computers were largely used for analyzing data and calculating mathematical problems. Yet, a little over 200 years ago, a precursor to the modern computer was used to create art and used punched cards to do so. The Jacquard Loom was a 2-story tall loom that was used to create elaborate weavings and used holes punched in cards in specific patterns and feeding those cards in a specific order to create those beautiful patterns.

Art and computers go way back. One of the earliest programming languages created for learning how to program was Logo, released in 1967.

Introduced with the Logo programming language, was the idea of a Turtle as a robot to draw out the programmed image. The image is programmed as a series of moves and turns with the "pen" up or down on the "paper" and drawn accordingly.



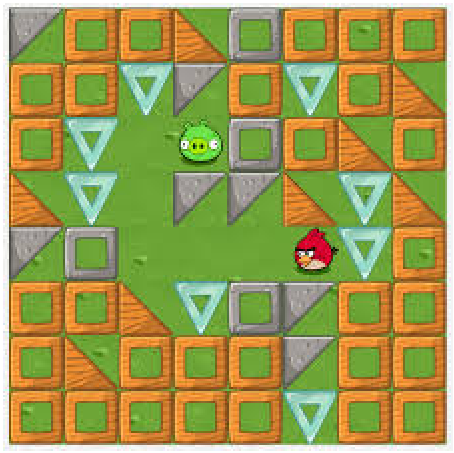


```
clean
home
repeat 72 [square right 5]
```

```
repeat 4 [forward 200 right 90]
```

Fig. 1: Image courtesy of Wikipedia. An animation that shows how the turtle is used to create graphics by combining forward and turn commands. (This is not actual Logo, but pseudo-code Logo.) CC by SA 4.0

Today, one might see the same kind of programming on the Code.org implementation of Hour of Code.



In Scratch, we can draw using a Logo-like syntax. If you imagine the screen as paper and the sprite as the pen, "pen up" and "pen down" are concepts that are not that far fetched. In Scratch, we have "pen" options like "erase all," "pen up," "pen down," and "pen color."

We can move a penned sprite 300 steps, turn 181 degrees, move 300 steps, and keep repeating. Example: Colorful Circle Using Straight Lines

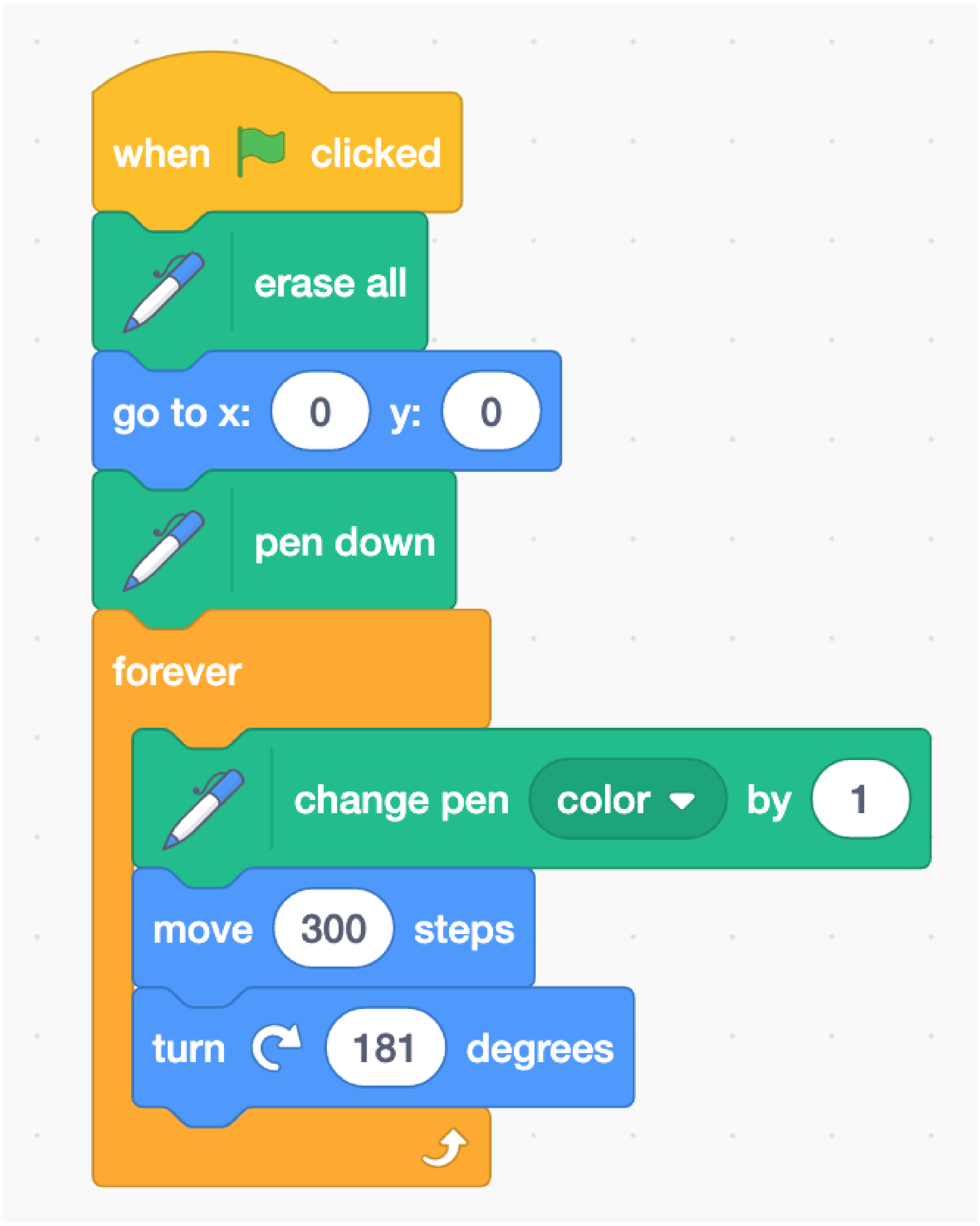
What do you think you can draw with a coded pen and a few simple commands?

1.1 Example Projects

- Scratch: Pen Experiment
- Scratch: Laser Cat
- Scratch: Colorful Circle Using Straight Lines

1.2 Project Ideas

- Draw polygons
- Random art generator
- Cycle game



```
when [flag] clicked
erase all
go to x: (0) y: (0)
pen down
forever
    change pen (color ▾) by (1)
    move (300) steps
    turn ↻ (181) degrees
```

CHAPTER

TWO

VARIABLES

When we talk with each other, we use variables all of the time. In the phrase, “Dasia played Minecraft yesterday; she thinks it is a cool game,” the word “she” is a variable that can mean “Dasia” (in another sentence mean someone completely different) and “it” is a variable that means “Minecraft.” We also talk about things that change as being “variable.” For instance, “the temperature this week has been variable, dropping below zero and rising above 50!”

In mathematics, we learn that a variable is something unknown that can take a value. In early math, you might see a number sentence like: $2+ = 5$; we know that $= 3$. Later on, that becomes $2 + x = 5$; we know that $x = 3$ in order for that math sentence to make sense.

In programming, variables can be something unknown that takes a value, as in the case of a variable that takes the value of something that the user inputs. A variable can also take on the value of something more complicated to make referencing that complicated thing more easy. This can be like setting a variable dojo to Kansas City instead of having to type out that word each time you want to use it.

A variable can also be changed; think about having a variable called “high score.”

In Scratch, you can create a data variable and give it a value. The variable can be used by either the sprite or all sprites. You can show your variable and its value on the stage if you want to; this would be useful for a game score.

Additional information about Scratch variables

In JavaScript, you can create a variable and give it a value.

```
// Note: each of these variable styles mean something different

var myCity = "Kansas City";

// or

let myCity = "Kansas City";

// or

const myCity = "Kansas City";
```

Additional information about JS variables

2.1 Example Projects

- Scratch: Number Guesser
- Scratch: Tony M. Bat-tie
- Scratch: Drew H. Space Game

- Website: Same G. Necrozma's UltraCalculator

2.2 Project Ideas

- Mad Libs
- Reaction timer that keeps track of the quickest time

CHAPTER

THREE

MAKING DECISIONS: IF-THEN-ELSE

When we are faced with a simple decision, how do we know which choice to make? Let's say that your family has all of its toothbrushes in the same cup in the bathroom. How do you know which is yours? Do you have the kid-sized toothbrush with the red handle or the one with the Spiderman handle?

When I was a kid and my mom yelled at me, "Eric, get in here." I knew that I had about 60 seconds to get to her; however, if she yelled, "Eric Aloysius, get in here right now," I knew, since she used my fake middle name, that I was in deep trouble and I had roughly 1.5 seconds to appear before her. In my head, I thought, "Given that mom yells my name. If she uses my middle name, I had better drop whatever I'm doing and run to her. Else, I can gently stop whatever I'm doing and quickly walk to her." So, when Mom yelled my name as "Eric Aloysius," did I walk to her or did I run to her?

In code, this might look like:

```
var name;

if (name === "Eric Aloysius") {
  me.dropEverything();
  me.runToMom();
} else {
  me.saveGameOrBookmarkBook();
  me.walkToMom();
}
```

You might notice in the code above, that what happens after I'm called depends on the truth of the name including my middle name or not. This kind of truth is called a "boolean." A boolean value is either true or false (also, yes or no). If-statements, such as the one above, rely on booleans. Let's look at another boolean.

Have you ever made a deal with your parents that if you get a certain grade or score on a test, you can celebrate by doing something special like going out for ice cream or pizza? Let's say that you were required to get a 90% or better on your test in order to celebrate. In this case, the boolean value is "score >= 90." Your score is either greater-than-or-equal-to 90, or it isn't.

```
var score;

if (score >= 90) {
  me.celebrate();
} else {
  me.scrubFloor();
}
```

If I were going to scrub the floor no matter what my score, my code might look a little different:

```
if (score >= 90) {
  me.celebrate();
```

(continues on next page)

(continued from previous page)

```
}

me.scrubFloor();
```

You can also use several if-statements together:

```
var score;

if (score === 100) {
  me.say("WOO!");
} else if (score >= 90) {
  me.say("Woo!");
} else if (score >= 80) {
  me.say("Woo.");
} else if (score >= 70) {
  me.say("ugh");
} else if (score >= 60) {
  me.say("eep!");
} else {
  me.say("Nonononononononono");
}
```

You can also check against several booleans (truths) together:

```
var score;
var subject;

if (score < 60 && subject === "spelling") {
  me.askTeacher("Do you offer test retakes?");
}
```

As you can see from these examples, there are several ways to use if-statements to make decisions and move your code along.

3.1 Example Projects

- Scratch: Number Guesser
- Scratch: Harmony M. Good or Bad
- Scratch: Tony M. Shark Bite

3.2 Project Ideas

- Ask the player for a favorite sport, if the sport is played professionally here, reply back with “Go, (name of team)!”
- Get the current temperature, if it’s colder than 60 F, show appropriate clothes for the temperature.

CHAPTER

FOUR

ARRAYS AND LISTS

Arrays are collections of things that belong together. Arrays are found all over math and science. We have collections of antennas, called "antenna arrays," that are used for listening to things far away.

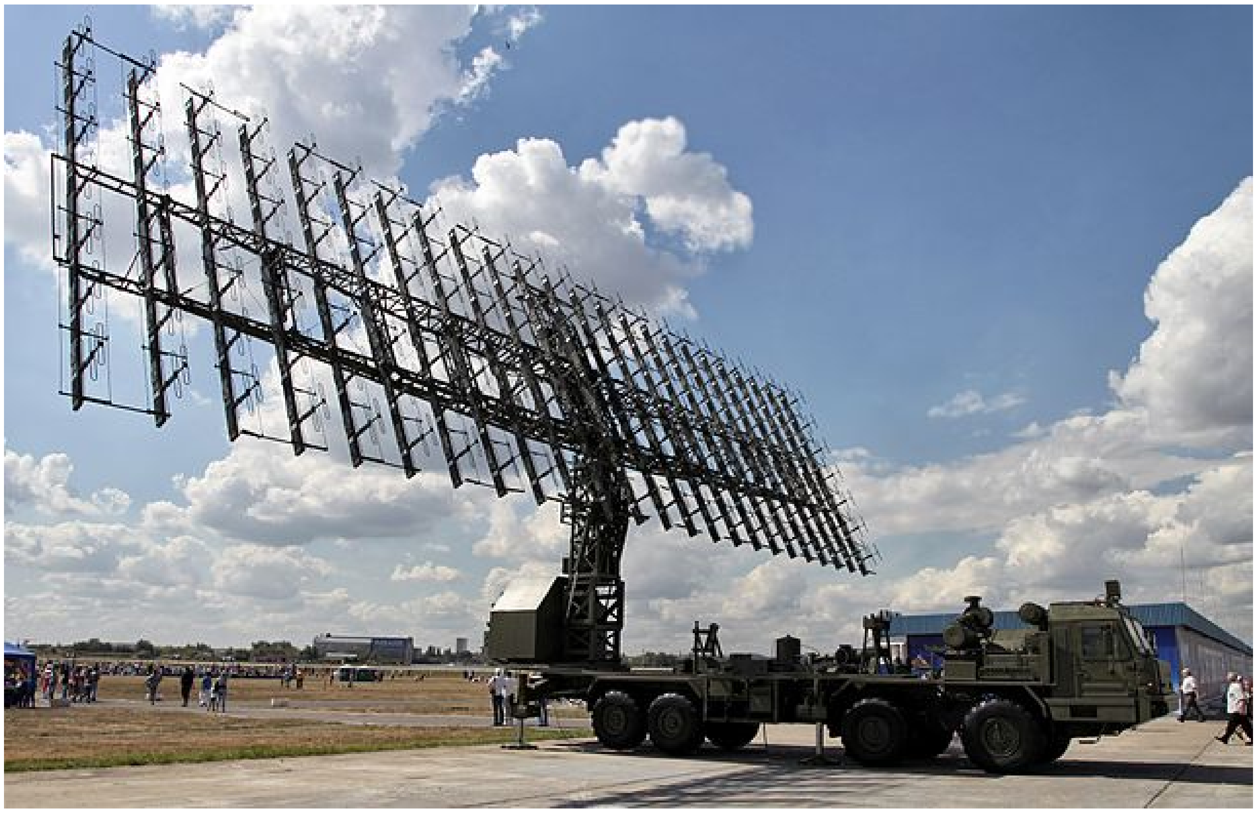


Fig. 1: Large planar array antenna of a VHF Russian mobile air defense radar, the Nebo-M. It consists of 175 folded dipole antennas driven in phase. It radiates a narrow beam of radio waves perpendicular to the antenna. By Vitaly V. Kuzmin, CC BY-SA 4.0

We have arrays of phone switches.

We have arrays of computer memory.

Kids who have worked with multiplication are introduced to arrays that look like this.

The items in your array belong together. Think of your kitchen as a collection of arrays. The refrigerator is an array of things that need to stay cool. The freezer is an array of things that need to stay frozen. The pantry is an array of food that can be room temperature. You have an array of plates, an array of cups and glasses, an array of eating utensils, an array of towels, an array of spices, etc.

Fig. 2: Array of telephone switches and their operators during WWII. Credit: U.S. National Archives

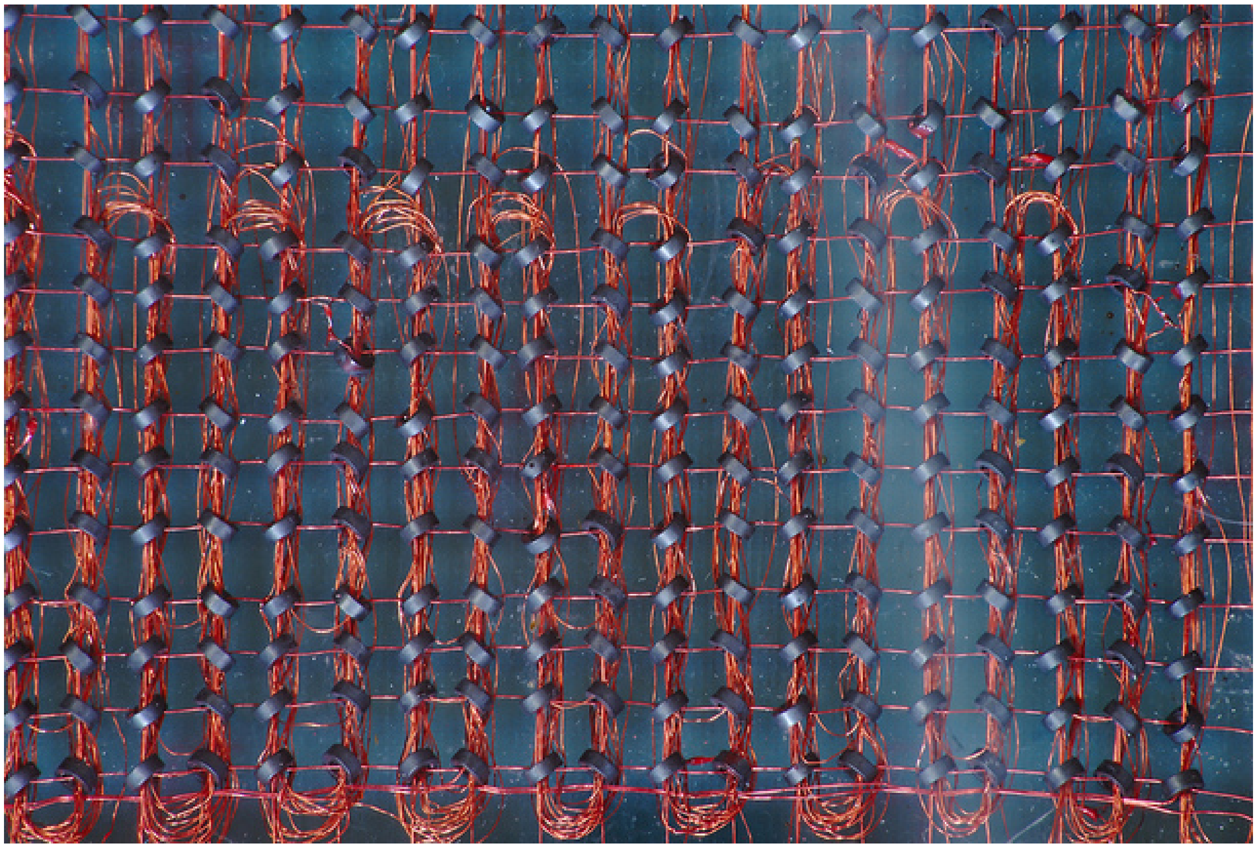


Fig. 3: Memory ferrite cores. Non-volatile computer memory with ferrite cores, invented in the late 1940s and used until the 1960s. By Frédéric BISSON, CC BY 2.0

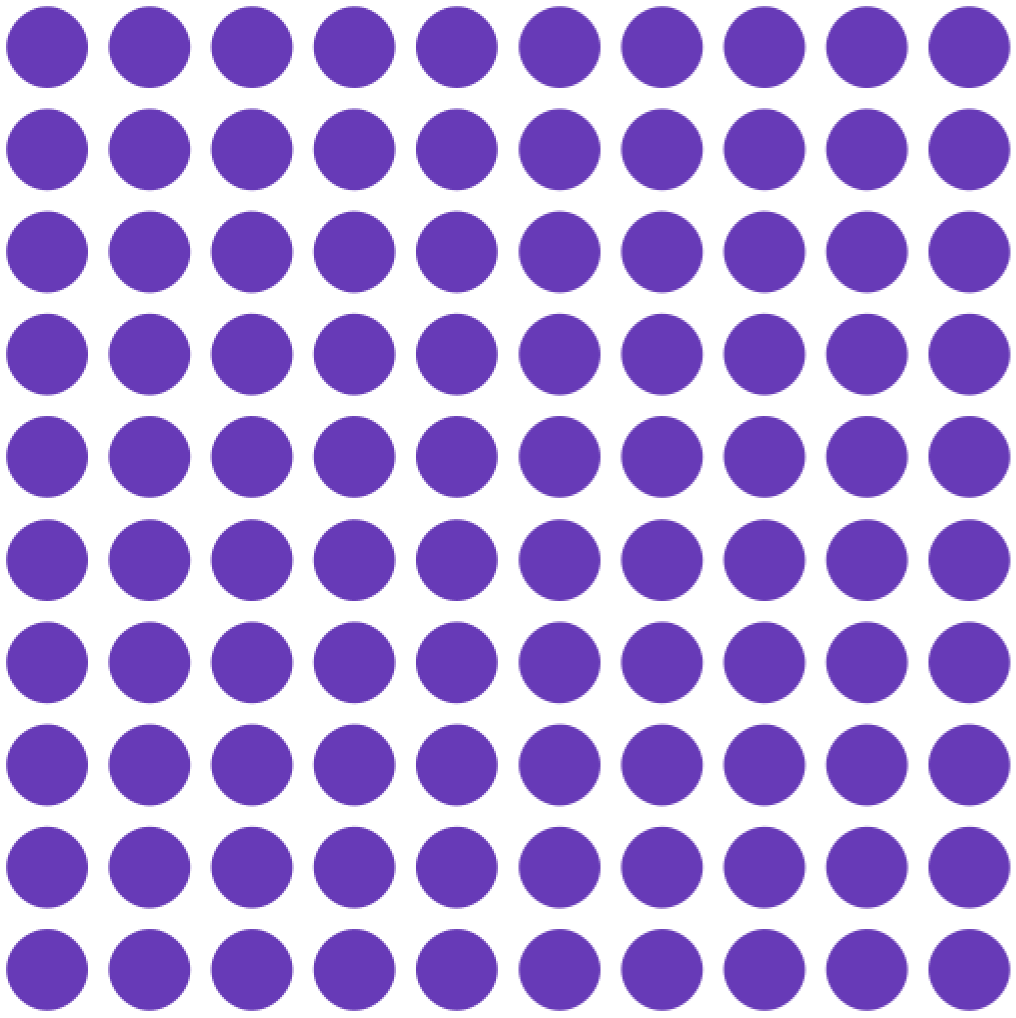


Fig. 4: Math array used for multiplication. Credit: SVG Silh, *CC0 1.0*

Arrays also have a size. In my array of cereal, I have 4 boxes of cereal. In my array of silverware, I have arrays of forks (8), salad forks (8), tea spoons (8), soup spoons (7), butter knives (8), and steak knives (8).

In simple arrays like above, each item is in a numbered slot called a "key" or an "index." In most programming languages, arrays start at zero (0) and count upward until you get to one less than the size of the array. So, for an array of 8 spoons, you would count them as "spoon 0, spoon 1, spoon 2, spoon 3, spoon 4, spoon 5, spoon 6, and spoon 7." In a Scratch list, you start counting at 1. This can get confusing, but it's important to understand if that's how items are arranged in the programming language that you are using, because you get the item in a simple array by its key number.

```
var movies = [
  "Black Panther",
  "Coco",
  "A Wrinkle in Time"
];

// the first movie in this list, "Black Panther," is:
var firstMovie = movies[0];
```

We can also have an array of arrays. This is called a "multi-dimensional array." Imagine a box of donuts. There are probably 12 donuts in that box of donuts. So, you might have:

```
var donuts = [
  "d1", "d2",  "d3",  "d4",
  "d5", "d6",  "d7",  "d8",
  "d9", "d10", "d11", "d12"
];
```

The third donut in that box is donuts[2] since we start counting in arrays at zero. But we know that we actually have 3 rows of 4 donuts in this box of donuts:

```
var donuts = [
  ["d1", "d2",  "d3",  "d4" ],
  ["d5", "d6",  "d7",  "d8" ],
  ["d9", "d10", "d11", "d12"]
];
```

Now, getting that third donut would require us to identify which row it's in, too: *donuts[0][2]*

Question: how do I get the last donut?

```
donuts[x][y]; // what is "x"? what is "y"?
```

Bonus Question: if I had 10 boxes of 12 donuts, how would I get the last donut?

```
donuts[x][y][z]; // what is "x"? "y"? "z"?
```

Bonus Question, just for Jenna:

```
// What did Hagrid say to:

[
    'Dumbledore',
    'McGonagall',
    'Snape',
    'Moody',
    'Tonks',
    'Black',
```

(continues on next page)

(continued from previous page)

```
    ...
]; // ?

console.log("You're a wizard array");
```

Scratch

See info about the Scratch version of an array, the list block.

4.1 Example Projects

- Scratch: My Favorite Movie
- Scratch: Adam P. Eevee Mixer
- Scratch: Aaron O. Asteroids

4.2 Project Ideas

- Mad Libs
- Inventory list in a game
- Last 3 high scores

CHAPTER

FIVE

LOOPS AND REPEATING THINGS

5.1 Introduction to Loops

Loops let you repeat something over and over again. For example:

5.1.1 Loop example 1

2015 had 12 months: January, February, March, April, May, June, July, August, September, October, November, December

2016 had 12 months: January, February, March, April, May, June, July, August, September, October, November, December

2017 had 12 months: January, February, March, April, May, June, July, August, September, October, November, December

2018 had 4 months: January, February, March, April, … which months are next?

5.1.2 Loop example 2

Spin around and count each spin until you are dizzy. How many times could you spin around?

5.2 Loops in Code

Your day could be condensed to "Wake up, do stuff, go to bed."

```
while (alive) {
    me.wakeUp();
    me.doStuff();
    me.goToBed();
}
```

In Scratch, this could look like:

```
// Scratch
(repeat until (not alive)
    (wakeUp)
    (doStuff)
    (goToBed)
)
```

You'll notice that even though these loops appear to go on forever, they have a conditional for when they can exit the loop. You want to have an "exit condition" to ensure that you don't have an endless loop in your program.

In JavaScript, the school year could be written as:

```
for (i = 0; i < 40; i++) {
    console.log("Let's go to school this week!");
}
// Let's go to school this week!
// Let's go to school this week!
// Let's go to school this week!
// Let's go to school this week!
// Let's go to school this week!
// ...
```

Here, the exit condition is that it has looped 40 times. It cannot loop 41 times.

You can also do something with the variables that are part of that loop.

```
var weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
↪"Saturday"];

for (i = 0; i < weekdays.length; i++ ) {
    console.log(weekdays[i]);
}

// Sunday
// Monday
// Tuesday
// Wednesday
// Thursday
// Friday
// Saturday
// Sunday
```

This could be rewritten using a special type of array loop called a `foreach`:

```
weekdays.forEach(function(day) {
    console.log(day);
});
```

What are some things that you can loop over? You can loop over items in a list, like repeating back the list of items you are supposed to get at the store.

You can also use a loop to repeat an action. For instance, what if I want to have a character walk across the screen? I could say:

```
// Scratch
(move(10))
(useNextCostume)
(wait(1))

(move(10))
(useNextCostume)
(wait(1))

(move(10))
(useNextCostume)
(wait(1))
```

(continues on next page)

(continued from previous page)

```

(move(10))
(useNextCostume)
(wait(1))

(move(10))
(useNextCostume)
(wait(1))
```

Or, I could put those actions in a loop:

```
// Scratch
(repeat(5)
    (move(10))
    (useNextCostume)
    (wait(1))
)
```

5.3 Infinite Loops: good and bad

There's a special kind of loop called an "Infinite Loop" (in Scratch: "Forever Loop"). This is a loop that is easy to create, usually accidentally, and difficult to get out of.

```
// Don't do this
var i = 0;
do {
    console.log(i++);
} while (i > 0);
```

```
// Don't do this, either
var i = 0;
do {
    console.log('I am awesome!');
} while (true);
```

So when is a good time to use an infinite loop? Perhaps while you are waiting for user input in a game or waiting for a status to change. In either case, you want to make sure that you have a way to exit the loop.

- *break:* - this exits a loop
- *continue:* - this skips the current step in the loop

```
while (true) {
    if (this.receivedQuitCommandFromUser()) {
        break;
    }

    this.playRelaxingMusic();
}
```

```
// Written as an infinite loop
printOddNumbersTo: function(topNumber) {
    for (var i = 1;; i++) {
        if (i % 2 === 1) {
```

(continues on next page)

(continued from previous page)

```
                continue;
            }

            // guard statement to get us out of the infinite-loop
            if (i >= topNumber) {
                break;
            }

            console.log(i);
        }
}
```

```
// Written as a non-infinite loop
printOddNumbersTo: function(topNumber) {
    for (var i = 1; i <= topNumber; i++) {
        if (i % 2 === 1) {
            continue;
        }

        console.log(i);
    }
}
```

5.4 Example Projects

- Scratch: Loop Through Names
- Scratch: Ethan G. Cup Game
- Scratch: Random Number Generator
- Scratch: Summer Plan: Go hiking! – forever-loops used for animation

5.5 Project Ideas

- Reduce repeated steps to show a character walking across the screen
- Read names in a list that start with the letter "R"

CHAPTER

SIX

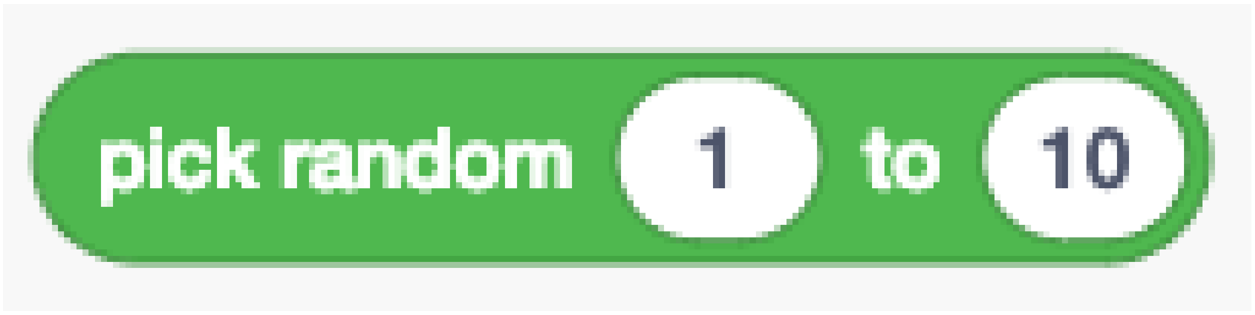
RANDOM

Let's take a look at a concept that helps make many games interesting, the concept of "random." If something is random, one has a difficult time predicting its behavior. Many games rely, in a huge part, upon randomness. For instance, in a card game, the dealer will randomize the deck of cards by shuffling the cards; which cards you get are selected from a randomly ordered deck. We roll dice to come up with a random number between 1 & 6. In American Football, a coin is tossed to determine which team starts the game by kicking or by receiving.

Randomness in Nature can be truly random. Computers, on the other hand, thrive on predictability and, therefore, making something truly random on a computer is very difficult. Here at the CoderDojo, we can attempt to use the computer's definition of random to our advantage to help make certain games more fun to play.

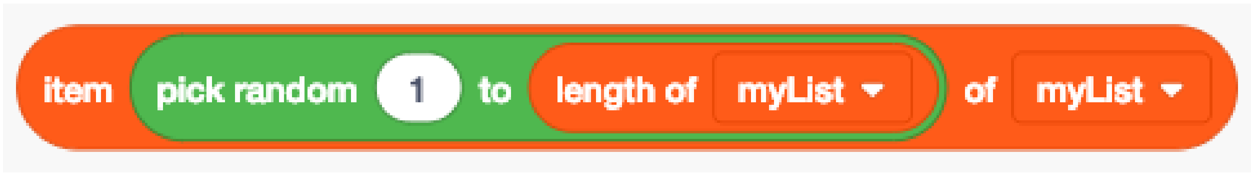
In Scratch, we have two ways of doing something randomly. We can pick a random number between a starting number and an end number.

```
(pick random (1) to (10))
```



We can use this to select a random item from a list:

```
(item (pick random (1) to (length of [myList]) of [myList])
```



In JavaScript, we generate a random number and use that to access an item in an array or to show a random number. Because JavaScript random numbers are a decimal between 0 and 1, we have to multiply that random number by however many numbers we want to select from and then round that number down to the nearest whole number. For example:

```
var randomize = function(things) {
    let randomNumber = Math.random() * things.length;
    let arrayItem = Math.floor(randomNumber);

    return things[arrayItem];
};
```

This function will take in an array and return a random item from that array.

6.1 Example Projects

- **Scratch: Smack Adam**
  - Check out the "Your" sprite for an example of how to got to a random location on the screen
- Scratch: Random Number Generator
- Scratch: Summer Plan: Go Hiking!

6.2 Project Ideas

- Have the target of your game appear in a random location on the screen
- Play a random note on an instrument
- Roll dice

CHAPTER

SEVEN

FUNCTIONS

Have you ever found yourself repeating your code over and over in Scratch or in JavaScript? If you find yourself doing this, you can take advantage of “functions” to reuse the same code over and over again. In Scratch, functions are called “my blocks,” but everywhere else, they’re called “functions” and sometimes “methods.”

All functions do something within your code.

Functions can take input and return output. This is like a popcorn popper at the movie theater: you add popcorn kernels and oil, and you get hot popcorn in return.

Other functions are like commands. They don’t take any input, but they do something anyway. It’s like telling your dog to sit. You don’t tell your dog where to sit, how to sit, how long to sit, or when to sit; you just tell your dog to sit. And guess what? Your dog sits. Good dog!

One way to think about functions is that they let you take a bunch of steps and give them a simple name. This is like teaching a robot to speak. You could tell the robot how to speak each time you want the robot to speak, or you can just create a speak function and tell the robot to do it instead.

```
voiceBox.turnOn();
voiceBox.setVolume(30);
voiceBox.say("Hello"); // This is the part that changes
voiceBox.turnOff();
```

```
voiceBox.turnOn();
voiceBox.setVolume(30);
voiceBox.say("I like to play games"); // This is the part that changes
voiceBox.turnOff();
```

The same, but with functions!

```
function speak(phrase) {
    voiceBox.turnOn();
    voiceBox.setVolume(30);
    voiceBox.say(phrase); // This is the part that relies on the input!
    voiceBox.turnOff();
}

speak("Hello");
speak("I like to play games");
```

As you can see, you can create the function once and call it multiple times. Notice, also, that you can have functions call functions, too!

You should also expect the same output for the same input in each function.

```
function double(thing) {
    return thing + thing;
}

let bigPrice = double(2); // output: 4
let tau = double(3.14); // output: 6.28
let freeKidsConferenceInKC = double("kc"); // output: "kckc"

// Check this out
function embiggen(thing) {
    return double(double(double(double(double(double(thing))))));
}

let bigNum = embiggen(2); // output: 128
```

Additional information about Scratch My Blocks Additional information about JS functions

7.1 Example Projects

- Scratch: Harmony M. & Adam P. The Busy Life of Business Cat
- Scratch: Tanner R. Taco Topper 1.0
- Scratch: Eric P. Froggy vs. Ravenous Flies

7.2 Project Ideas

- Dance choreography, with each move being a function that describes that move
- A demo of how to cook your favorite meal

CHAPTER

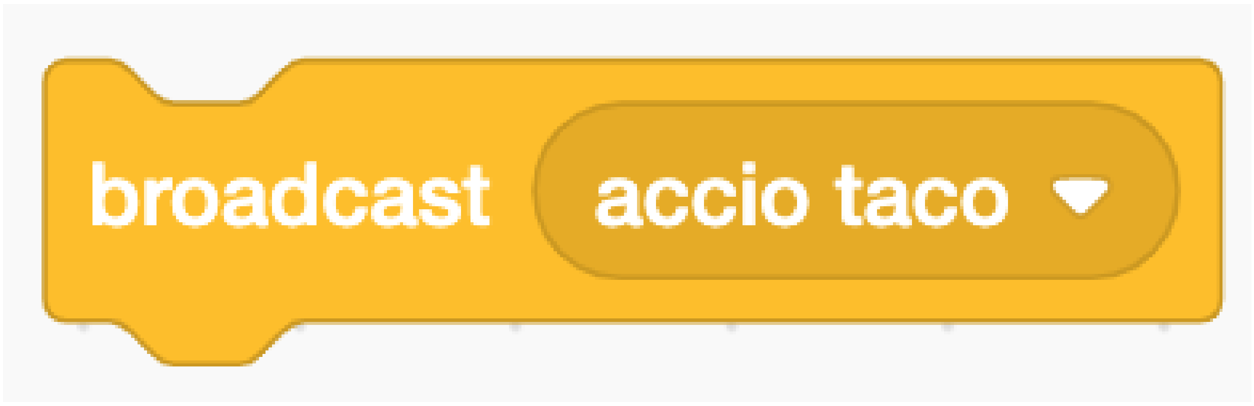
EIGHT

BROADCAST/RECEIVE EVENTS

Today we are going to flex our code-magic muscles and look at events, broadcast/receive (aka publish/subscribe). On a web page or in a game, clicking on a button requires an action or an event (let's call it "on-click"); that action will broadcast a message (ex. "HEY! The red button was clicked!") that another part of the program is listening for. Once that message is received, an action will take place (ex. "OK, I now know that the red button was clicked, so I will ignite the rocket"). In Scratch, you would use the "Events" blocks for "broadcast" and "receive." In JavaScript, you might use the event for "onclick" and run a function.

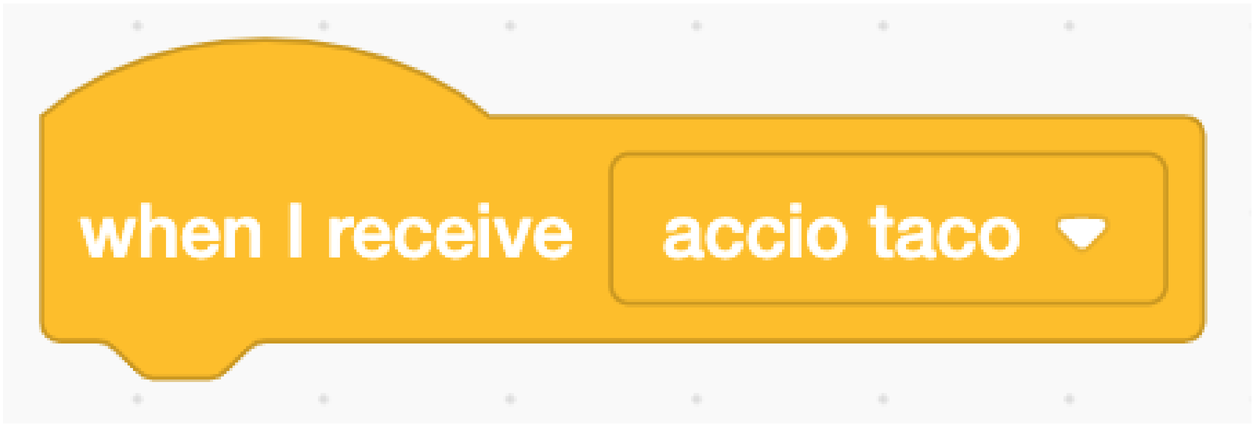
In a way, this is like casting a spell. You say the words or make the right motion and the appropriate action takes place. I could say "Accio taco!" and wave my arms just so and a taco from my favorite taco stand would fly into my hands. "

"Accio taco" is the Scratch Broadcast or the JavaScript event.



broadcast accio taco

The taco flying to me would be the Scratch "when I receive" or the JavaScript function that is called when the event happens.



when I receive accio taco

8.1 Example Projects

- Scratch: Summon Taco
- Scratch: KP Goal
- JavaScript: Events Garden

8.2 Project Ideas

- Tell a story with 2 characters talking back and forth
- Create a Rube Goldberg Machine where the completion of each step is a broadcast to tell the next step to begin.
- Build a sound-effects machine where each button is associated with a different sound.

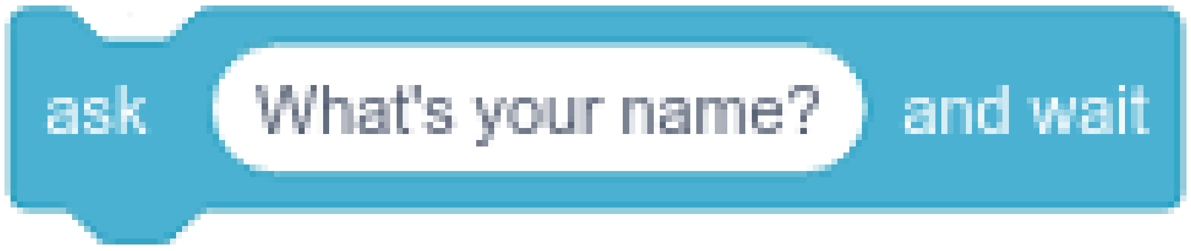
CHAPTER

NINE

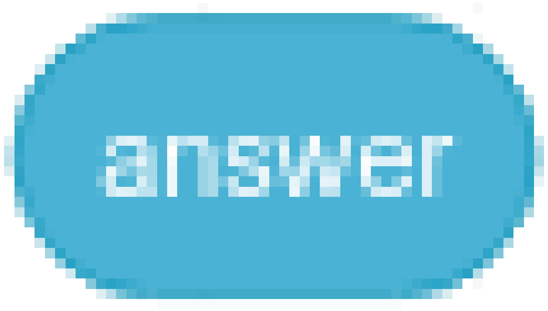
USER INPUT: TEXT

If you ask almost any developer-mentor in the CoderDojo who they write programs for, they will all tell you that they write them for clients, other people. All of these programs rely upon input from those other people in order to be useful. If you think about the games that you love to play, those games are fun to play because they rely upon some kind of input from you — it could be the press of a button, wiggling a joystick, or typing an answer. Today, we'll look at the a user typing in text in order to provide input for your program.

In Scratch, we have one way of getting user input as text:

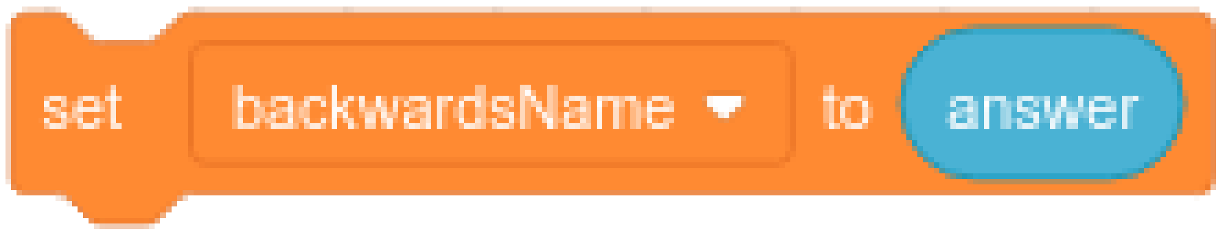


This will ask for input and store that input as an answer variable that you can use later.



If you ask several questions, you will want to store each of those answers in a different variable before asking another question.

In JavaScript, you can get user input by using a form. A form can take in text, ask questions where the user has to select either "yes" or "no," offer a dropdown list for users to select one or more options, and others. Each form part should have a unique ID since that the answer to that part of the form will be tied to the ID.



```
<input type="text" id="place">
<button type="submit" id="submit">Submit</button>
```

In JavaScript, you would then pull information out of that form to be used in your code. You can do that by assigning the data tied to those form part IDs to variables.

```
var place = document.querySelector("#place");
```

You can then show that answer in a string by using string concatenation

```
"I love " + place.value + ", it is such a cool place to visit!"
```

The Mozilla Developer Network has a great introduction to forms if you'd like to learn more about creating forms in HTML

9.1 Example Projects

- Scratch: Your Name Backwards
- Scratch: Yusuf S. AMERICA

9.2 Project Ideas

- Mad Libs
- calculator
- flash cards
- trivia game

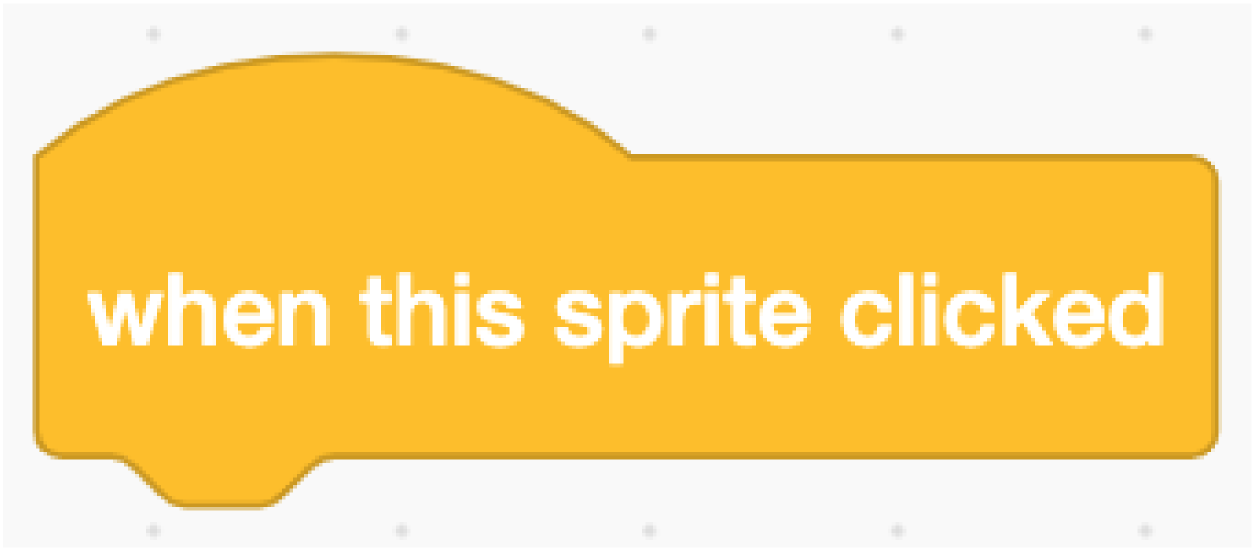
CHAPTER

TEN

USER INPUT: BUTTONS

If you ask almost any developer-mentor in the CoderDojo who they write programs for, they will all tell you that they write them for clients, other people. All of these programs rely upon input from those other people in order to be useful. If you think about the games that you love to play, those games are fun to play because they rely upon some kind of input from you — it could be the press of a button, wiggling a joystick, or typing an answer. Today, we'll look at the a user selecting an option in order to provide input for your program.

In Scratch, we have one way of getting user input as a selection, in the form of a button:



This will fire an event that you can act upon. Whatever code is attached to this block will run when this sprite is clicked.

In JavaScript, you can get user input by using a form. A form can take in text, ask questions where the user has to select either "yes" or "no," offer a dropdown list for users to select one or more options, and others. Each form part should have a unique ID since that the answer to that part of the form will be tied to the ID.

```
<input type="text" id="place">
<button type="submit" id="submit">Submit</button>
```

In JavaScript, you can get user input by using a form. A form can take in text, ask questions where the user has to select either "yes" or "no," offer a dropdown list for users to select one or more options, and others. Each form part should have a unique ID since that the answer to that part of the form will be tied to the ID.

```
<input type="radio" id="firstGroup" value="2019">
    October 14, 2019
```

(continues on next page)

(continued from previous page)

```
</input>
<button type="submit" id="submit">Submit</button>
```

In JavaScript, you would then pull information out of that form to be used in your code. You can do that by assigning the data tied to those form group IDs to variables.

```
var answer = document.querySelector("#firstGroup");

//You can then check the answer by using an if-statement.
if (answer == "2019") {
    doSomething();
}
```

The Mozilla Developer Network has a great introduction to forms if you'd like to learn more about creating forms in HTML.

You can also have your buttons do interesting things. For instance, you can have buttons change your background color.

```
<button id="red" onclick="redder()">More Red</button>
```

```
var red = 0;
var green = 0;
var blue = 0;

function updateBackground(red, green, blue) {
    document
        .querySelector('body')
        .style.backgroundColor
            = "rgb(" + red + "," + green + "," + blue + ")";
}

function redder() {
    this.red = this.red <= 250 ? this.red + 10 : 0;

    this.updateBackground(this.red, this.green, this.blue);
}
```

10.1 Example Projects

- Scratch: Magic School
- Scratch: Lucas F. Windows 9 1.1
- Scratch: John M. Button Blast

10.2 Project Ideas

- calculator
- Simon
- Jumping game