3.2.4 Add Combatants to the Battle Using Arrays

With only one robot to fight, winning isn't exactly a challenge. To impress the game jam judges, we'll need to add some difficulty to Robot Gladiators. Let's do that by adding some more enemy robots to the mix.

But before we do that, let's pause the `fight()` function. We're not currently working on this function, and those pesky alert dialogs can get a bit tiresome.

PAUSE

How do you stop the `fight()` function from executing?

Comment out the `fight()` function call at the bottom of the `game.js` file.

Hide Answer

We can comment out a line or multiple lines of code by adding `//` to the beginning:

```
//fight()
```

When you comment out a line, the font turns green and the command will not execute. There is no need to comment the function expression, because it only defines the function. The function doesn't actually execute until the last line in the `game.js` file (in the `fight()` function call).

Okay, now we're ready to add some more combatants. To start, let's focus on the "WIN" game state.

One way to do this would be to declare each enemy robot name in the `game.js` file, like this:

```
var enemy1 = "Roborto";
var enemy2 = "Amy Android";
var enemy3 = "Robo Trumble";
```

This wouldn't be so bad if we were only introducing a couple of robots. But what if we later decide to add 100 robots or more? Typing out each enemy's variable and expression would be time-consuming and add bloat to the codebase. So how can we avoid that terrible scenario?

This is where we introduce a special new data structure called an **array**. Unlike the data types we've seen before, arrays are a data type in JavaScript designed to store other data as a list, and it is what we'll use to create a list of enemies. So how does an array differ from the types of data we've used before?

In JavaScript, there are two types of data: Primitives and Objects. Primitive data is data that only holds one value, such as a string, number, or boolean. Objects are a looser term for a data type that can store more than one value of data, arrays being one of them. We'll learn more about the different object data types as we go, but let's take a minute and watch this video explaining what primitive data types are before we move on:

The last step of declaring this array is to remove any other `enemyName` or `enemyName1` declarations. Going forward, we'll only use the array for enemy name storage.

Access the Array

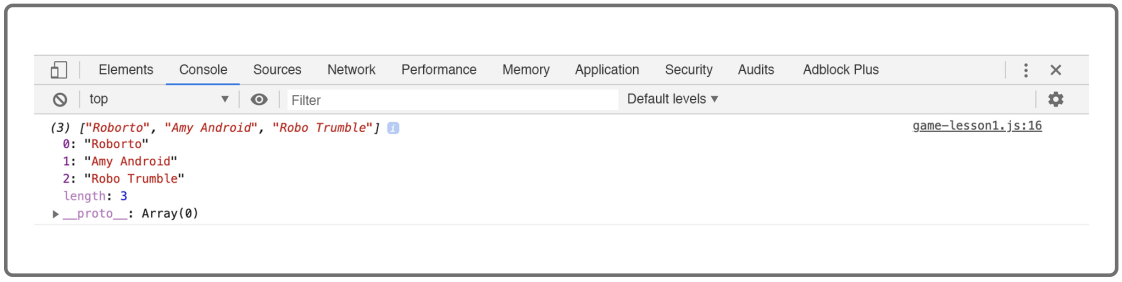
So how do we retrieve the values (aka our enemy combatants) once we've stored them in the array? Array elements are stored in the array at specific indexes. These indexes act like the tabs that help organize the contents of a file cabinet. When a particular index is called, the value stored at that index is retrieved. The first index in an array is zero, and the indexes increment by one for each corresponding element. So the second element will be at index 1, the third element will be at index 2, and so on.

> **NERD NOTE**
>
> Arrays that start their index at zero are called **zero-indexed arrays**. These are the most common arrays in programming.

To display the array in the console and review the results, type the following statement beneath the array declaration in the `game.js` file:

```
console.log(enemyNames);
```

Save the file, then refresh the browser. In the console you will see the array displayed. If you click on the arrow to the left of it, it will expand as illustrated in the following image:



To the left of the array is a number. Can you guess what this number represents?

This number (in this case, 3) is the length of the array, which has three elements. Beneath the number is a list of the array elements. The number next to each element represents the index of the element in the array. Again, the first index starts with zero, and the indexes increment by one for every following element.

IMPORTANT

> Keep the console window in the browser open during the development process so you can see the errors and results displayed there.

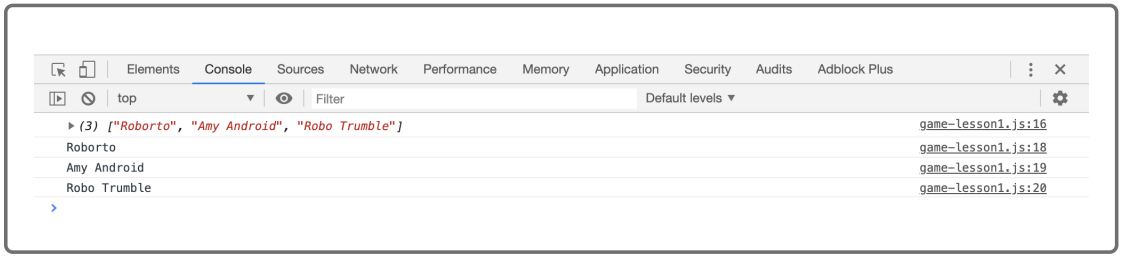
To retrieve the first element in the array, we'll use the following statement:

```
enemyNames[0];
```

In this line of code, we called the array and used the bracket syntax with the element's index number. The browser can interpret this notation to retrieve the element in the array at the zero index.

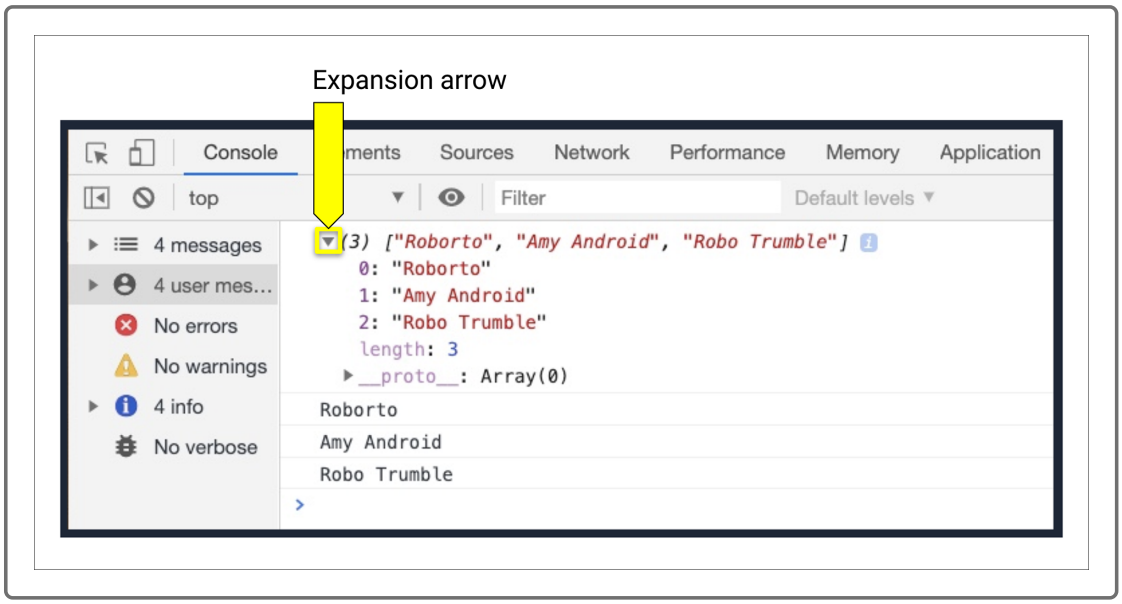
Try to display each element of the array in the console by typing the statements into the `game.js` file beneath the array declaration.

Your result in the console should look like this:



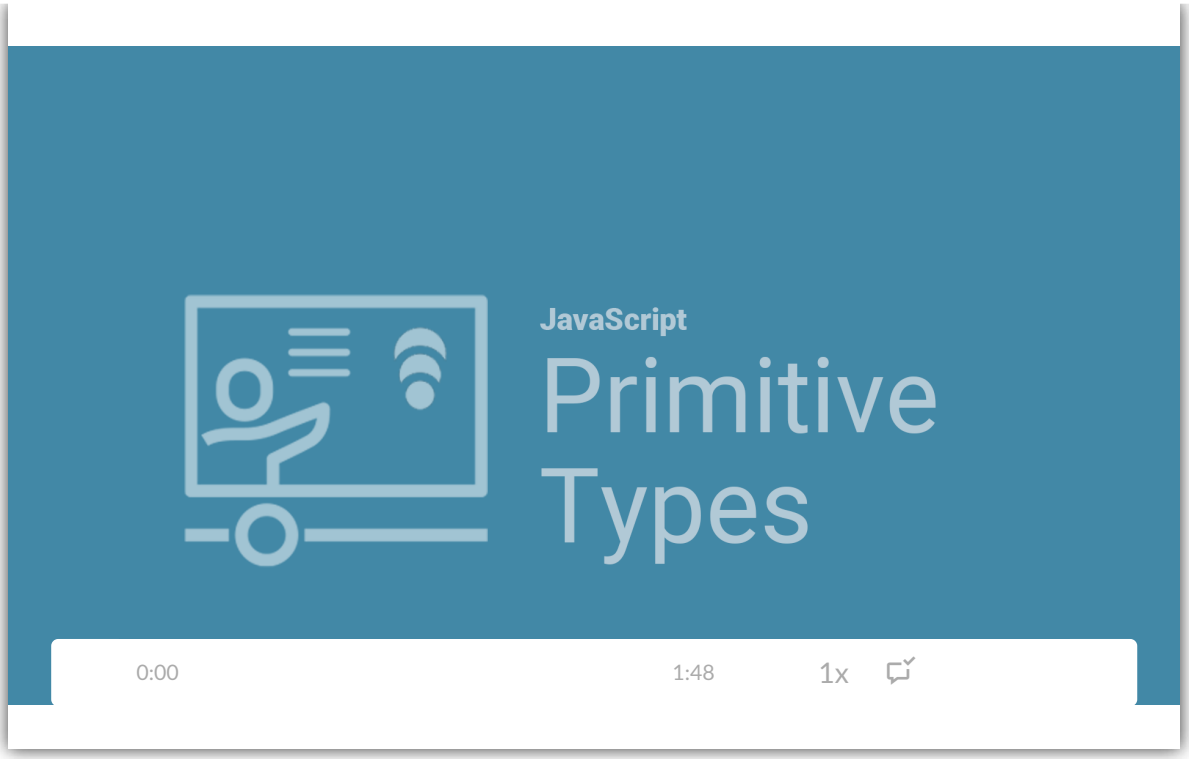
Notice how each array element is displayed with the file name and line number of each statement on the right side of the console window (e.g., `game-lesson.js:16`). This identifies which `console.log` statement is being displayed.

We can also expand the array in the console window by selecting the arrow to the right of the array as displayed in the image below:



In the image above, the value and the index number of the value in the array are displayed along with the length property of the array and the number of elements or the length of the array value. The `_proto_` property will be explained in a future lesson and is not in scope of this lesson, but feel free to investigate on your own.

Type the following statement in the `game.js` file and explain the result:



Declare an Array of Enemy Combatants

Let's create our first array with the enemy robots, then go over some of the properties that arrays offer. Replace the `enemyName` declaration near the top of the `game.js` file with the following expression:

```
var enemyNames = ["Roboto", "Amy Android", "Robo Trumble"];
```

We now have three enemy combatants in a array! How did we do this? To declare an array, we use the `var` keyword followed by the array name. We assign the array using brackets `[ ]` that contain the array elements. **Array elements** are the values stored within the array. In our case, the `enemyNames` array contains three array elements, which are strings, separated by commas.

Array elements can be other data types as well, including numbers, Booleans, variables, and even objects. But for the purpose of our game, an array can act as a single repository that stores all the enemy robot names.

```
console.log(enemyNames[3]);
```

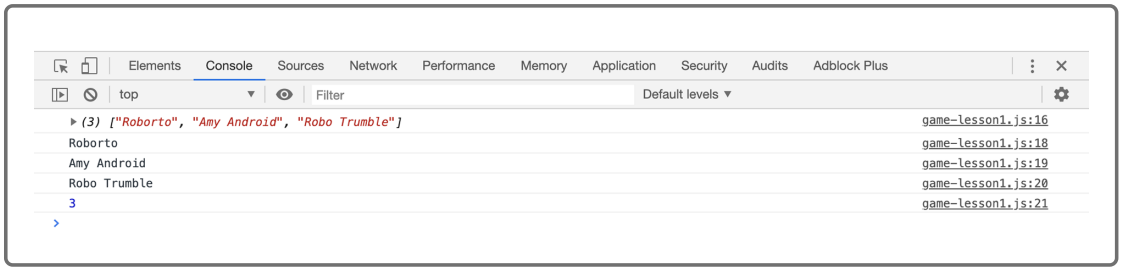
Note how this statement displays as **undefined**. This is because the last element is at index 2, so there is no element in the array at index 3. If we try to access an array at an index that doesn't contain an element, the result will be undefined because this element hasn't been declared yet.

A JavaScript array is actually a type of object, so it has some useful built-in properties and methods. For example, the `length` property of the `Array` object contains the number of elements in the array.

To use it to find the length of our array, change the `console.log()` statement in `game.js` to the following:

```
console.log(enemyNames.length);
```

In the console, we can see that the `length` property displays the number 3, which is the length of our array:



PAUSE

How can we use the `length` property to find the last element in the `enemyNames` array?

`enemyNames[enemyNames.length - 1]` will return the last element of an array. We subtract 1 from the total length to find the last

index because the first index of an array is zero.

Hide Answer

Now that we've added our array of enemy combatants, let's see what we can do with them.