3.2.5 Fight Enemy Robot Combatants Using For Loops

In the prior example, to display each element of the array in the console, we had to write out each element and its corresponding index:

```
console.log(enemyNames[0]);
console.log(enemyNames[1]);
console.log(enemyNames[2]);
```

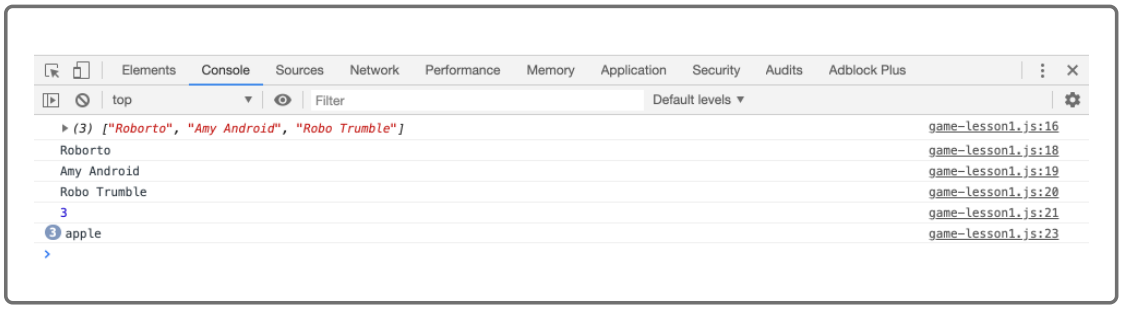
Just imagine if we wanted to display each element in an array containing over 100 items. This would be incredibly time-consuming and repetitive.

Luckily, we have a statement that enables us to loop through the array, no matter the length: the `for` loop. This statement can be used whenever we need the same operation done repetitively, so it's not just for arrays. Let's first take a general approach, then we'll see why arrays go with `for` loops like peanut butter goes with jelly!

In this example, imagine that we would like to display the word "apple" three times in the console. Type the following `for` loop in the `game.js` file beneath the `console.log(enemyNames.length)` line:

```
for(var i = 0; i < 3; i++) {
  console.log("apple");
}
```

We should see the following in the console window:



Notice the number 3 next to the word "apple" in the console. This means that this word was displayed three times in the console. Great—the `for` loop worked!

Let's break down the syntax and see what makes the `for` loop special.

For Loop Syntax

The `for` loop is a special type of statement called a **control flow**. The control flow is the order in which the computer executes statements in a JavaScript file or script, which normally runs sequentially from the first line to the last line. Control flow statements, such as conditional statements or `for` statements, change the control flow based on the statement's conditions.

DEEP DIVE

DEEP DIVE

Still not sure what the differences are between statements, expressions, and functions? To learn more, see this **StackOverflow article on the difference between statements and functions (https://stackoverflow.com/questions/9307291/difference-between-statement-and-function)** .

Just like `if` statements, `for` statements also have conditions:

```
for([initial expression]; [condition]; [increment expression]) {
   statement
}
```

When a `for` loop executes, the following occurs:

1. The **initial expression** is the first statement executed, initializing the loop iterator or counter. This expression can also declare variables.

   Here's an example of an initial expression:

   ```
   var i = 0;
   ```

2. The **condition** statement is then evaluated. If this condition evaluates to true, the loop **statement** executes. If the condition's value is false, the `for` loop terminates. If this condition is omitted, the condition is assumed to be true.

   Here's an example of a condition:

```
i < 3;
```

3. Then the **statement** executes. To execute multiple statements, use a block statement `{ }`, as used in `if-else` statements.

4. The **increment expression** executes, incrementing the iterator, which is the variable `i`.

   Here's an example of an `IncrementExpression`:

   ```
   i++
   ```

   This is equivalent to `i = i + 1`.

5. After the iterator increments, the **condition** is then reevaluated and continues the loop chain.

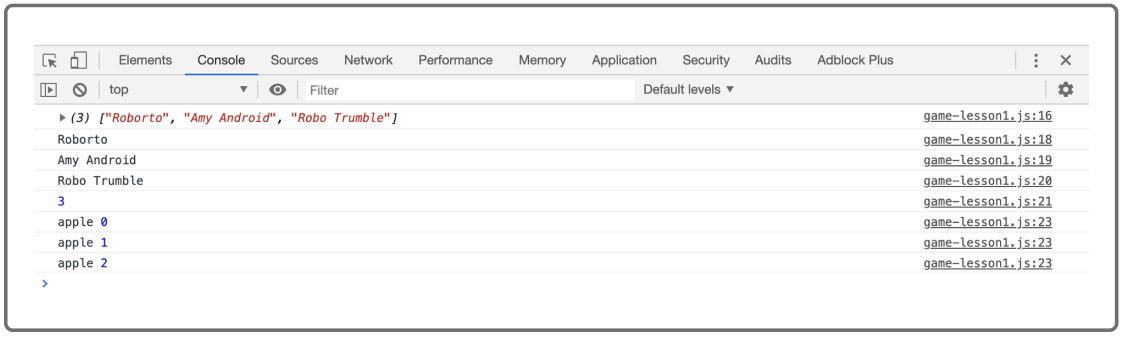
To see the iterator in action, let's display it in each loop by adding an argument in the `console.log` function with a comma:

```
for(var i = 0; i < 3; i++) {
  console.log("apple", i);
}
```

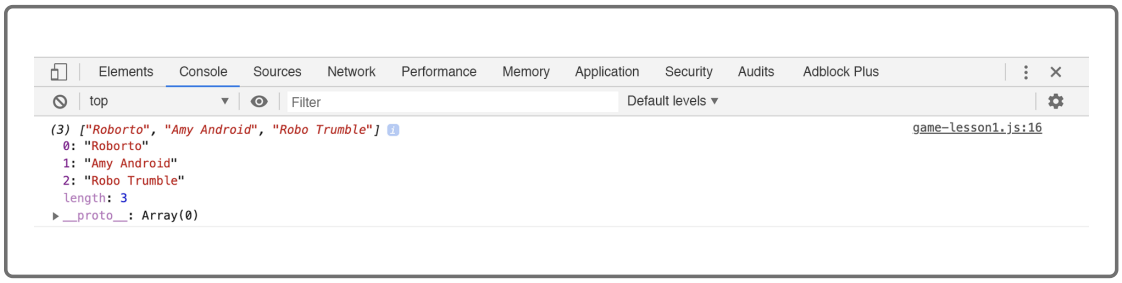
Alternatively, we could concatenate the string and the variable as follows:

```
console.log("apple " + i)
```

We should see the following in the console:



Notice how the iterator starts at zero and ends at 2, like the first array we examined in the console:



We can actually use the iterator in the `for` loop to map to an array's index. This allows us to iterate through an array, element by element, index by index.

For Loops and Arrays

Let's examine how to use a `for` loop through an array to display each element. Replace the previous `for` loop with the following statement:

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

We should get the following in the browser:

Due to the increasing number of results we are displaying in the console, it may become difficult to recognize which results we just created. Reviewing the line numbers to the right of the displayed values pinpoints which line of code created the displayed result. For instance, the last three entries we created with the `for` loop maps perfectly to the `for` loop in the `game.js` file.

Congratulations, you just looped through your first array!

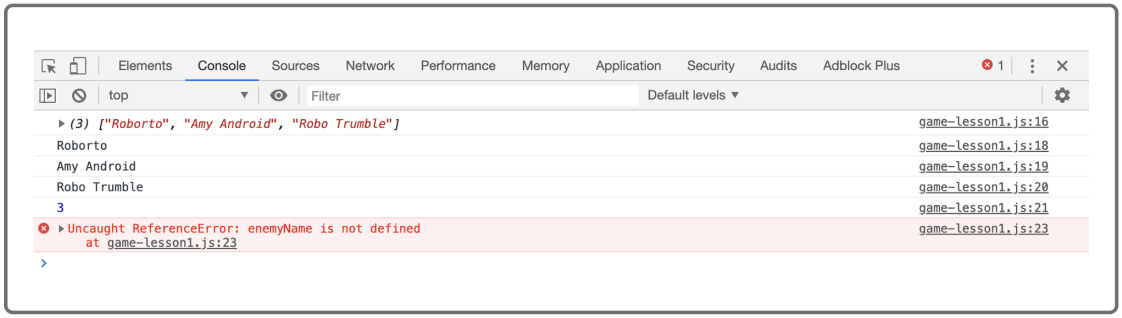
Replace the `for` loop with the following code:

```
for(var i = 0; i < enemyNames.length; i++) {
  console.log(enemyNames[i]);
  console.log(i);
  console.log(enemyName[i] + " is at " + i + " index");
}
```

Did you notice the bug in this code? Save `game.js` and refresh the browser. What happened in the console?

The Console Is Your Friend

The browser will notify you in the console if you have errors in your code. For instance, here is a common error that every developer becomes very familiar with after a short time programming:



The error is `Uncaught ReferenceError: enemyName is not defined`.

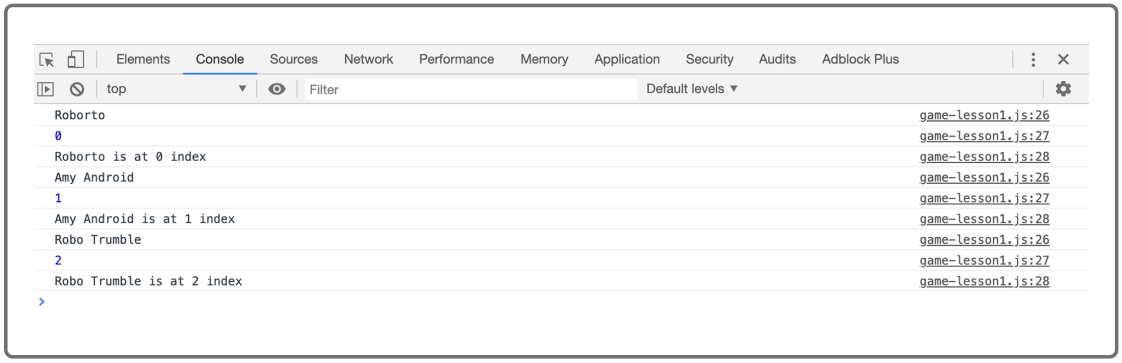
This message from the browser states that the variable `enemyName` is being used but has never been declared, so the browser can't execute the statement. Again, it gives us the file name and line number where we can find this error. It looks as though we mistakenly tried to reference `enemyName` as the array in the `for` loop instead of `enemyNames`. Typos like this often trigger errors.

HIDE PRO TIP

Cultivate the mindset that console errors are helpful indicators. Carefully reading error messages can help you rectify mistakes more quickly than if you just skim over them.

Let's fix the code and execute the program again.

We should see the following in the console:



Much better! Now that we know how to loop through the `enemyNames` array, let's see how we can apply these operations to achieve the following pseudocoding objectives:

```
// Game States
// "WIN" - Player robot has defeated all enemy robots
//    * Fight all enemy robots
```