3.1.6 Manage Data with Functions, Variables, and Operators

While the last step focused on functions, variables, and putting them both to use, this step will be all about manipulating data using operators.

By the end of this step, we'll be able to execute our `fight()` function, have our robot attack an enemy, and have the enemy attack our robot. Let's start by asking ourselves what we need in order to make this happen.

For our robots to battle each other, they will require some type of data tracking of their overall health values as the game progresses. We'll also need to know the attack power of each robot to determine how much damage it can do to its opponent. This sounds like a job for variables, so let's update our `game.js` file. While we're at it, we'll give the enemy robot a name.

Replace everything in `game.js` so that the entire file looks like this:

```
var playerName = window.prompt("What is your robot's name?");
var playerHealth = 100;
var playerAttack = 10;

// You can also log multiple values at once like this
console.log(playerName, playerAttack, playerHealth);
```

```
var enemyName = "Roboto";
var enemyHealth = 50;
var enemyAttack = 12;

var fight = function() {
  window.alert("Welcome to Robot Gladiators!");
};

fight();
```

We've now set up our game to keep track of our player robot's information and the enemy robot's information, each receiving three variables to hold their names, health points, and attack points.

Save `game.js` but before you refresh `index.html` in your browser, try to predict what will happen when you do! Don't forget to look at the DevTools console.

PAUSE

What type of data does the `enemyHealth` variable hold? What about `enemyName`?

The `enemyHealth` variable is a Number data type, and the `enemyName` variable is a String data type.

Hide Answer

We also made a slight change to our `fight` function. Let's explore that.

There are two methods to create a function in JavaScript:

- **Function declaration:** This is when we create a function using the `function` keyword first. Here's an example:

```
// create function
function fight() {
  window.alert("Welcome to Robot Gladiators!");
}
// execute function
fight();
```

- **Function expression:** This is when we create a function by assigning it to a variable. Here's an example:

```
// create function
var fight = function() {
  window.alert("Welcome to Robot Gladiators!");
};
// execute function
fight();
```

Both methods are acceptable when it comes to creating functions in JavaScript, and you'll see them written both ways throughout your career. But for the sake of consistency, we'll use the second method and employ function expressions throughout our game. There is one main difference in how the browser interprets those two methods, but we won't tackle that issue just yet since it won't affect us throughout the game jam.

Do Math with Operators

Now we have the important data needed for our robots to battle, so let's update the `fight` function to actually make it happen. We'll update the function to do five things when we execute it:

1. Alert users that they're starting the round (this is already done).
2. Subtract the value of `playerAttack` from the value of `enemyHealth`, and use that result to update the value in the `enemyHealth` variable.

3. Log a resulting message to the console to confirm that it worked.

4. Subtract the value of `enemyAttack` from the value of `playerHealth`, and use that result to update the value in the `playerHealth` variable.

5. Log a resulting message to the console to confirm that it worked.

Think about the above list as a logical outline of the actions our function needs to perform. Before we create any function, no matter how simple, we should always consider what needs to be done and write it out in our own words.

We can put this list of actions into our code as comments, to remind us of the order in which we need our code to run and to remind us what it's doing.

Let's update the `fight` function to look like this:

```
var fight = function() {
  // Alert users that they are starting the round
  window.alert("Welcome to Robot Gladiators!");

  //Subtract the value of `playerAttack` from the value of `enemyHealt

  // Log a resulting message to the console so we know that it worked.

  // Subtract the value of `enemyAttack` from the value of `playerHeal

  // Log a resulting message to the console so we know that it worked.
};
```

Now we have a set of instructions in our code, so let's tackle the next two steps and learn about operators.

Let's add the following code to our `fight` function, below each comment:

```
//Subtract the value of `playerAttack` from the value of `enemyHealth`
enemyHealth = enemyHealth - playerAttack;

// Log a resulting message to the console so we know that it worked.
console.log(
  playerName + " attacked " + enemyName + ". " + enemyName + " now has
);
```

Once these are in place, save the file and refresh the browser.

HIDE HINT

Make sure the Chrome DevTools Console stays open while you work on our game! It'll refresh itself when you refresh the page.

We can see in the console that something has happened to the value of our enemy's health. It seems to be less than what we set it to when we created the `enemyHealth` variable. We just manipulated and reassigned a variable's value!

Let's dissect the code to better understand what happened, starting with the first line:

```
enemyHealth = enemyHealth - playerAttack;
```

First we can see that there is no `var` keyword here. We use `var` to create new variables, but in this case, we want to update the value of a variable that already exists. Using the `var` keyword would create a new `enemyHealth` variable inside the `fight` function, meaning that the `enemyHealth` variable we created at the top of the file and outside the

function would be unaffected. This is called **scoping** a variable—more on this later.

On the left of our expression, we simply list the variable we're updating and storing new data to. We update it by reassigning it a new value, determined by the following code:

```
enemyHealth - playerAttack;
```

The above code is on the right side of our expression. This may seem confusing—how can we set the `enemyHealth` variable to `enemyHealth - playerAttack`? Aren't we using the same name (`enemyHealth`) twice? Well, yes, but variable names have different meanings to the left or right of an assignment operator. Listing the variable on the left side means we'll store data to that variable, and listing the variable on the right side means we'll use the actual value that variable holds at that moment.

After the attack finishes, we use `console.log()` to give ourselves some details as to what just happened. This is a good way to confirm that our code is working. If we called the `fight` function again, the second `console.log()` would show that the value of `enemyHealth` is ten points lower than the previous log.

Now that we have our robot attacking the enemy robot correctly, let's put our new knowledge to use by having the enemy robot attack our robot.

Add the following into the `fight` function in the `game.js` file below each comment:

```
// Subtract the value of `enemyAttack` from the value of `playerHealth
playerHealth = playerHealth - enemyAttack;

// Log a resulting message to the console so we know that it worked.
console.log(
  enemyName + " attacked " + playerName + ". " + playerName + " now ha
);
```

We just repeated the previous two lines of code and switched the variable names around! When you save `game.js` and reload `index.html`, you'll see that we have a full working round of battle between our robots, because they can exchange attacks.

DEEP DIVE

DEEP DIVE

Arithmetic operators in JavaScript work much like they do in everyday math, with one important exception.

JavaScript uses the **addition operator** (`+`) not only to produce a sum of two or more numbers but also to concatenate strings. A string data type on either side of the `+` operator will result in the two values forming a string instead of being added together as a sum:

```
// this will be 10
console.log(4 + 6);

// this will be 46
console.log("4" + 6);
```

The rest of the arithmetic operators behave the same way that you learned in math class. For more information, see the **MDN web docs on operators. (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators)**

Our `fight()` function is taking shape, as we can now have our robots attack each other. But how will the game know when to end?

The game should end when one of the robots' health points reduce to zero, so we have to instruct our code to check for that. More importantly, we have to tell our code what to do when that happens. Let's work on that next.