3.2.6 Fight Each Combatant Using Function Arguments

Now that we can access each enemy robot in the `enemyNames` array, we must try to have our robot fight them in the `fight()` function. Currently a global variable named `enemyName` inside the `fight()` function makes the robots face off and fight each other. Now that we have an `enemyNames` array, we'll need to change how the robot combatants face off.

Let's take a minute to think about a scenario in which we make the robots fight as we loop through the `enemyNames` array. Because the `fight()` is a function, we can call it inside the `for` loop. Now the use of functions becomes more apparent! We can repeatedly execute this code in the `fight()` function from within a loop.

But how do we pass the enemy robot into the function? Presently the function expression can only fight a single robot. We'll use a property of functions that passes data into the function through an **argument**. This method allows us to pass a new enemy robot into the `fight()` function on every loop iteration:

```
fight(enemyRobot);
```

We've used arguments whenever we've passed information into a function, whether it be a variable or a value. Here are some examples:

```
console.log(enemyNames);      // array argument
alert("Hello");               // string argument
console.log(enemyNames[i], i); // two arguments, comma-separated
```

We've been using arguments with `window` methods and JavaScript functions, but now we'll change the `fight()` function to pass in an argument. Remember, the function expression defines the "how" and "what" of the function's operations. If we want the function to perform differently, we need to change the function's definition. In this case, we want to convert the `fight()` function to pass in an enemy robot and then fight this robot.

Let's find the `fight()` expression and change the first line to the following:

```
var fight = function(enemyName) {
  ... // fight function statements
}
```

We just redefined the `fight` variable to a function that can input or receive a variable. Because we're dealing with a variable placeholder in the function definition, we call this the function's **parameter**.

DEEP DIVE

DEEP DIVE

Parameters are often confused with arguments because their syntax is similar. The main distinction between them is their purpose in the function. In the following function expression, a parameter serves as a variable placeholder that indicates how the variable will be used in the function. Because the parameter is only used for the scope of the function, the name of the parameter isn't significant, but it normally relates to the purpose of the variable, as follows:

```
var wash = function(soapType) {
 console.log("I wash with " + soapType);
};
```

When we call the `wash()` function, we can enter a type of soap into the argument. The argument is used when a function is called with a value as an input.

```
wash("Irish Spring"); // => I wash with Irish Spring
```

This will be displayed in the console.

As part of the function call, the argument passes information into the function—whether that be a variable or a value—with any data type or structure.

Let's look in the `fight()` function and notice the changes created by adding `enemyName` as the parameter. Now `enemyName` is defined by the value we pass to this argument at the function call. Notice that `enemyName` maps perfectly within the function to the original `enemyName` variable that we used to store the enemy robot's name.

Organizing the Application

Now that the `fight()` function has been modified, let's continue with our goal to make our robot fight each of the enemy robots. Before we proceed, let's make sure the app is correctly organized. The global variables should be at the top of the `game.js` file, along with the `enemyNames` array:

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

var enemyNames = ["Roborto", "Amy Android", "Robo Trumble"];
var enemyHealth = 50;
var enemyAttack = 12;
```

We removed the `console.log` statements for now to declutter the console window. The `fight()` function expression should then come next. This expression has to be above the `for` loop because we'll be calling the `fight()` within the loop.

If the function isn't declared before the function call, we'll get an error like the one we received earlier—with some differences:

`Uncaught TypeError: fight is not a function`

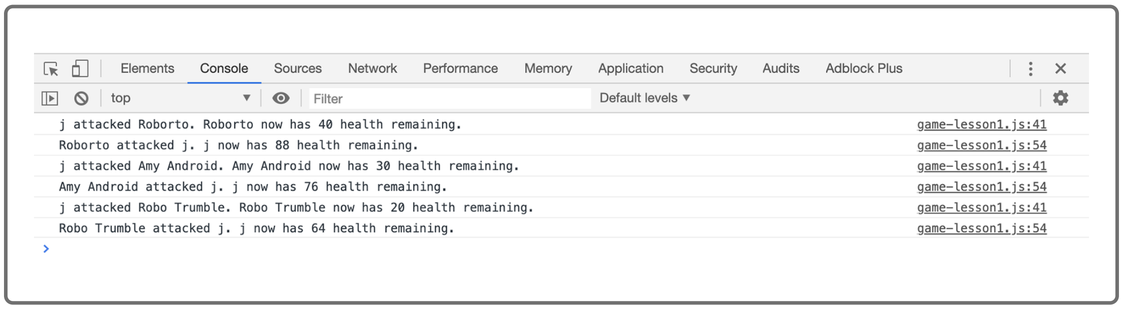
Now let's type the `for` loop into the `game.js` file, replacing the `fight()` function call at the bottom of the `game.js` file:

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

Notice that the `fight()` function call was replaced with a `for` loop that calls the `fight()` function multiple times using the element in the array `enemyNames[i]` as the argument.

As the `for` loop iterates through the array, the `fight()` function call, passes each enemy robot name into the `fight()` function. This completes our goal to face each robot in a battle round.

Save your work and reload the `index.html` file in the browser to demonstrate the application so far. We will get a result in the console panel similar to the image below:



```
j attacked Roborto. Roborto now has 40 health remaining.              game-lesson1.js:41
Roborto attacked j. j now has 88 health remaining.                    game-lesson1.js:54
j attacked Amy Android. Amy Android now has 30 health remaining.      game-lesson1.js:41
Amy Android attacked j. j now has 76 health remaining.                game-lesson1.js:54
j attacked Robo Trumble. Robo Trumble now has 20 health remaining.    game-lesson1.js:41
Robo Trumble attacked j. j now has 64 health remaining.               game-lesson1.js:54
>
```

As the console shows, we've successfully looped through the array and fought each robot. We've almost reached the MVP and successfully accomplished our objective to face and fight each enemy robot. The next goal is to fight multiple rounds with each enemy robot, deplete their health resources, and defeat them. Then we'll be able to "WIN" the game.

This is a great spot to add, commit, and push our work to GitHub for safekeeping.