LEARN BASIC JAVASCRIPT BY BUILDING A ROLE PLAYING GAME

Introduction:

JavaScript is a powerful scripting language that you can use to make web pages interactive. It's one of the core technologies of the web, along with HTML and CSS. All modern browsers support JavaScript.

In this practice project, you'll learn fundamental programming concepts in JavaScript by coding your own Role Playing Game. You'll learn how to work with arrays, strings, objects, functions, loops, `if/else` statements, and more.

Step 1:

JavaScript is a powerful language which allows you to build websites that are interactive.

Note: For all remaining projects in this curriculum, you will need a basic level of knowledge in HTML and CSS. If you are new to HTML and CSS, please go through the Responsive Web Design Certification.

To get started, create your standard HTML boilerplate with a `DOCTYPE`, `html`, `head`, and `body`, then add a `meta` tag for the `charset`. Add a `title` element and use the text `RPG - Dragon Repeller` for it. Include a `link` tag for your stylesheet to link the `styles.css` file.

Finally, create a `div` element with `id` set to `game` within your `body`.

Step 2:

Now you can start writing your JavaScript. Begin by creating a `script` element. This element is used to load JavaScript into your HTML file.

Example Code:

```
<script>

  // JavaScript code goes here

</script>
```

Step 3:

One of the most powerful tools is your developer console. Depending on your browser, this might be opened by pressing `F12` or `Ctrl+Shift+I`. On Mac, you can press `Option + ⌘ + C` and select "Console". You can also click the "Console" button above the preview window to see our built-in console.

The developer console will include errors that are produced by your code, but you can also use it to see values of variables in your code, which is helpful for debugging.

Add a `console.log("Hello World");` line between your `script` tags. Then click the "Console" button to open the console. You should see the text `"Hello World"`.

Step 4:

Before you start writing your project code, you should move it to its own file to keep things organized.

Remove your `console.log("Hello World");` line. Then give your now empty `script` element a `src` attribute set to `./script.js`.

Step 5:

Your view has been switched to your new `script.js` file. Remember that you can use the tabs above to switch between files.

Add your `console.log("Hello World");` line to this file, and see it appear in your console.

Step 6:

Remove your `console.log("Hello World");` line to begin writing your project code.

In the previous project, you learned how to work with variables and the `let` keyword like this:

Example Code:

```
let camperbot;
```

You also learned how to initialize a variable with a value like this:

Example Code:

```
let age = 32;
```

Use the `let` keyword to declare a variable called `xp` and assign it the value of `0`, a number.

Step 7:

Initialize another variable called `health` with a value of `100`, and a variable called `gold` with a value of `50`.

Step 8:

Create another variable called `currentWeaponIndex` and set it to `0`.

Step 9:

Declare a variable called `fighting` but do not initialize it with a value.

Step 10:

Declare two more variables named `monsterHealth` and `inventory`.

For your `inventory` variable, assign it the value of an array containing the string `"stick"`.

Remember that you worked with arrays in the previous project like this:

Example Code:

```
let exampleArray = ["first", "second", "third"];
```

Step 11:

In your role playing game, users will be able to track their stats, buy weapons, and fight monsters. Before you can continue with the interactive JavaScript portion of the game, you need to first create the HTML elements that will display the game information.

Create four `div` elements within your `#game` element. Give them the following respective `id` values, in order: `stats`, `controls`, `monsterStats`, and `text`.

Step 12:

Create three span elements within your #stats element. Give each of them the class stat. Then give the first one the text XP: 0, the second one the text Health: 100, and the third one the text Gold: 50.

Step 13:

Wrap the numbers 0, 100, and 50 in span elements, and wrap those new span elements in strong elements. Then give your new span elements id values of xpText, healthText, and goldText, respectively.

Your answer should follow this basic structure:

Example Code:

```
<span class="stat">TEXT <strong><span id="VALUE">TEXT</span></strong>
```

Step 14:

For your #controls element, create three button elements. The first should have the id set to button1, and the text Go to store. The second should have the id set to button2, and the text Go to cave. The third should have the id set to button3, and the text Fight dragon.

Step 15:

JavaScript interacts with the HTML using the Document Object Model, or DOM. The DOM is a tree of objects that represents the HTML. You can access the HTML using the document object, which represents your entire HTML document.

One method for finding specific elements in your HTML is using the querySelector() method. The querySelector() method takes a CSS selector as an argument and returns the first element that matches that selector. For example, to find the <h1> element in your HTML, you would write:

Example Code:

```
let h1 = document.querySelector("h1");
```

Note that `h1` is a string and matches the CSS selector you would use.

Create a `button1` variable and use `querySelector()` to assign it your element with the `id` of `button1`. Remember that CSS `id` selectors are prefixed with a `#`.

Step 16:

We have run into a slight problem. You are trying to query your page for a button element, but your `script` tag is in the `head` of your HTML. This means your code runs before the browser has finished reading the HTML, and your `document.querySelector()` will not see the button - because the browser hasn't processed it yet.

To fix this, move your `script` element out of the `head` element, and place it at the end of your `body` element (just before the closing `</body>` tag).

Step 17:

`button1` is a variable that is not going to be reassigned. If you are not going to assign a new value to a variable, it is best practice to use the `const` keyword to declare it instead of the `let` keyword. This will tell JavaScript to throw an error if you accidentally reassign it.

Change your `button1` variable to be declared with the `const` keyword.

Step 18:

Use querySelector() to get the other two button elements using their ids: button2 and button3. Store them in variables called button2 and button3. Remember to use const.

Step 19:

Similar to your #stats element, your #monsterStats element needs two span elements. Give them the class stat and give the first element the text Monster Name: and the second the text Health: . After the text in each, add a strong element with an empty nested span element. Give the first inner span element an id of monsterName and the second inner span element an id of monsterHealth.

Step 20:

Give your #text element the following text:

Example Code:

Welcome to Dragon Repeller. You must defeat the dragon that is preventing people from leaving the town.

You are in the town square. Where do you want to go? Use the buttons above.

Step 21:

Now we need some quick styling. Start by giving the body a background-color set to #0a0a23.

Step 22:

Give the #text element a background-color of #0a0a23, a color of #ffffff, and 10px of padding on all sides.

Step 23:

Give your #game a maximum width of 500px and a maximum height of 400px. Set the background-color to #ffffff and the color to #ffffff.

Use margins to center it by setting the top margin to 30px, bottom margin to 0px, and the left and right margin to auto.

Finally, give it 10px of padding on all four sides.

Step 24:

Using a selector list (selector1, selector2) give both your #controls and #stats elements a border of 1px solid #0a0a23, a #0a0a23 text color, and 5px of padding.

Step 25:

Give your #monsterStats element the same border and padding as your #stats element. Set its color to #ffffff and give it a #c70d0d background.

Step 26:

For now, hide your #monsterStats element with the display property. Do not change any of the other styling.

Step 27:

Next, give your .stat elements a padding-right of 10px.

Step 28:

Finally, you will need to add some styles for your buttons. Start by setting the `cursor` property to `pointer`. Then set the text `color` to `#0a0a23` and the `background-color` to `#feac32`.

Then set the `background-image` property to `linear-gradient(#fecc4c, #ffac33)`. Lastly, set the `border` to `3px solid #feac32`.

Step 29:

Just like you did with the buttons, create variables for the following `id`s and use `querySelector()` to give them the element as a value:

`text`, `xpText`, `healthText`, `goldText`, `monsterStats`, and `monsterName`.

Remember to declare these with the `const` keyword, and name the variables to match the `id`s.

Step 30:

Finally, use `querySelector()` to get the `#monsterHealth` element. Because you have already declared a `monsterHealth` variable earlier, you need to use a different variable name for this element.

Declare a new variable with the `const` keyword and name it `monsterHealthText`.

Step 31:

In the previous project, you learned how to create a function like this:

Example Code:

```
function functionName() {
```

```
}
```

Create an empty function named `goStore`.

Step 32:

For now, make your `goStore` function output the message `"Going to store."` to the console.

Step 33:

Now create a `goCave` function that prints `"Going to cave."` to the console.

Step 34:

Now create a `fightDragon` function that prints `"Fighting dragon."` to the console.

Step 35:

In the previous project, you learned how to work with single line and multi-line comments like this:

Example Code:

```
// I am a single-line comment

/*

  I am a multi-line comment

*/
```

Add a single-line comment that says initialize buttons.

Step 36:

button1 represents your first button element. These elements have a special property called onclick, which you can use to determine what happens when someone clicks that button.

You can access properties in JavaScript a couple of different ways. The first is with dot notation. Here is an example of using dot notation to set the onclick property of a button to a function reference.

Example Code:

```
button.onclick = myFunction;
```

In this example, button is the button element, and myFunction is a reference to a function. When the button is clicked, myFunction will be called.

Use dot notation to set the onclick property of your button1 to the function reference of goStore. Note that button1 is already declared, so you don't need to use let or const.

Step 37:

Using the same syntax, set the onclick properties of button2 and button3 to goCave and fightDragon respectively.

Once you have done that, open your console and try clicking the buttons on your project.

Step 38:

The innerText property controls the text that appears in an HTML element. For example:

Example Code:

```
<p id="info">Demo content</p>
```

Example Code:

```
const info = document.querySelector("#info");

info.innerText = "Hello World";
```

The following example would change the text of the p element from Demo content to Hello World.

When a player clicks your Go to store button, you want to change the buttons and text. Remove the code inside the goStore function and add a line that updates the text of button1 to say "Buy 10 health (10 gold)".

Step 39:

Now, add a line that updates the text of button2 to say "Buy weapon (30 gold)" and update the text of button3 to say "Go to town square".

Step 40:

You will also need to update the functions that run when the buttons are clicked again.

In your goStore() function, update the onclick property for each button to run buyHealth, buyWeapon, and goTown, respectively.

Step 41:

Now you need to modify your display text. Change the `innerText` property of the `text` variable to be `"You enter the store."`.

Step 42:

Create three new empty functions called `buyHealth`, `buyWeapon`, and `goTown`.

Step 43:

Move your `goTown` function above your `goStore` function. Then copy and paste the contents of the `goStore` function into the `goTown` function.

Step 44:

In your `goTown` function, change your `button` elements' `innerText` properties to be `"Go to store"`, `"Go to cave"`, and `"Fight dragon"`. Update your `onclick` properties to be `goStore`, `goCave`, and `fightDragon`, respectively.

Finally, update `innerText` property of your `text` to be `"You are in the town square. You see a sign that says Store."`.

Step 45:

You need to wrap the text `Store` in double quotes. Because your string is already wrapped in double quotes, you'll need to escape the quotes around `Store`. You can escape them with a backslash `\`. Here is an example:

Example Code:

```
const escapedString = "Naomi likes to play \"Zelda\" sometimes.";
```

Wrap the text `Store` in double quotes within your `text.innerText` line.

Step 46:

You have repetition in the `goTown` and `goStore` functions. Repetition in your code is a sign that you need another function.

In the previous project, you learned how to work with function parameters like this:

Example Code:

```
function myFunction(param) {

  console.log(param);

}
```

Function parameters act as placeholders for values that you pass to the function when you call it.

Create an empty `update` function that takes a parameter called `location`.

Step 47:

In your role playing game, you will be able to visit different locations like the **store**, the **cave**, and the **town square**. You will need to create a data structure that will hold the different locations.

Use `const` to create a variable called `locations` and assign it an empty array.

Step 48:

Before you can begin to build out your `locations` array, you will first need to learn about objects. Objects are an important data type in JavaScript. The next few steps will be dedicated to learning about them so you will better understand how to apply them in your project.

Objects are non primitive data types that store key-value pairs. Non primitive data types are mutable data types that are not `undefined`, `null`, `boolean`, `number`, `string`, or `symbol`. Mutable means that the data can be changed after it is created.

Here is the basic syntax for an object:

Example Code:

```
{
  key: value
}
```

You will learn about keys and values in the next few steps.

For now, create a `const` variable called `cat` and assign it an empty object `{}`.

Below that `cat` variable, add a `console.log(cat)` statement to see the object in the console.

Step 49:

Objects are similar to `arrays`, except that instead of using indexes to access and modify their data, you access the data in objects through `properties`.

Properties consist of a key and a value. The key is the name of the property, and the value is the data stored in the property.

Here is an example of an object with a single property:

Example Code:

```
const obj = {
  name: "Quincy Larson"
};
```

Inside your `cat` object, add a new property. The key should be `name` and the value should be the string `"Whiskers"`.

Open up the console to see the updates to your object.

Step 50:

If the property name (key) of an object has a space in it, you will need to use single or double quotes around the name.

Here is an example of an object with a property name that has a space:

Example Code:

```
const spaceObj = {
  "Space Name": "Kirk",
};
```

If you tried to write a key without the quotes, it would throw an error:

Example Code:

```
const spaceObj = {

  // Throws an error

  Space Name: "Kirk",

};
```

Add a new property with a key of `"Number of legs"` and value of `4` to the `cat` object.

Open up the console to see the output.

Step 51:

There are two ways to access the properties of an object: dot notation (`.`) and bracket notation (`[]`), similar to an array.

Dot notation is what you use when you know the name of the property you're trying to access ahead of time.

Example Code:

```
object.property;
```

Here is a sample of using dot notation (`.`) to read the `name` property of the `developer` object:

Example Code:

```
const developer = {

  name: "Jessica",

}

// Output: Jessica
```

```
console.log(developer.name);
```

Update your console statement to access the `name` property of the `cat` object using dot notation.

Open up the console to see the `name` of `"Whiskers"` logged to the console.

Step 52:

The second way to access the properties of an object is bracket notation (`[]`). If the property of the object you are trying to access has a space in its name, you will need to use bracket notation.

Example Code:

```
objectName["property name"];
```

Here is a sample of using bracket notation to read an object's property:

Example Code:

```
const spaceObj = {

  "Space Name": "Kirk",

};

spaceObj["Space Name"]; // "Kirk"
```

Update your console statement to use bracket notation to access the property `"Number of legs"` of the `cat` object.

Open up the console to see the output.

Step 53:

Later on in the curriculum, you will dive deeper into objects. But for now, it is time to apply what you have learned to your role playing game.

Start by deleting your `cat` object and console statement.

Step 54:

Your locations `array` will hold different locations like the **store**, the **cave**, and the **town square**. Each location will be represented as an object.

Inside your locations array, add an object. Inside that object add a key called `name` with a value of `"town square"`.

Remember to follow this syntax:

Example Code:

```
{
  key: value
}
```

Step 55:

Just like array values, object properties are separated by a comma. Add a comma after your `name` property and add a `button text` property with the value of an empty array.

Since the property name has a space in it, you will need to surround it with quotes.

Example Code:

```
{
  name: "Naomi",
  "favorite color": "purple"
}
```

Step 56:

Give your empty `button text` array three string elements. Use the three strings being assigned to the button `innerText` properties in the `goTown` function. Remember that array values are separated by commas.

Step 57:

Create another property in your object called `button functions`. Give this property an array containing the three functions assigned to the `onclick` properties in the `goTown` function. Remember that these functions are variables, not strings, and should not be wrapped in quotes.

Step 58:

Add one final property to the object named `text`. Give this property the same string value as the one assigned to `text.innerText` in the `goTown` function.

Step 59:

Add a second object to your `locations` array (remember to separate them with a comma). Following the pattern you used in the first object, create the same properties but use the values from the `goStore` function. Set the `name` property to `store`.

Step 60:

Now you can consolidate some of your code. Start by copying the code from inside the goTown function and paste it into your update function. Then, remove all the code from inside the goTown and goStore functions.

Step 61:

Instead of assigning the innerText and onclick properties to specific strings and functions, the update function will use data from the location that is passed into it. First, that data needs to be passed.

Inside the goTown function, call the update function. Here is an example of calling a function named myFunction:

Example Code:

```
myFunction();
```

Step 62:

Now it is time to use your update function. Pass in your locations array into the update function call.

You pass arguments by including them within the parentheses of the function call. For example, calling myFunction with an arg argument would look like:

Example Code:

```
myFunction(arg)
```

Pass your locations array into the update call.

Step 63:

The `locations` array contains two locations: the `"town square"` and the `"store"`. Currently you are passing that entire array into the `update` function.

Pass in only the first element of the `locations` array by adding `[0]` at the end of the variable. For example: `myFunction(arg[0]);`.

This is called bracket notation. Values in an array are accessed by index. Indices are numerical values and start at `0` - this is called zero-based indexing. `arg[0]` would be the first element in the `arg` array.

Step 64:

Now your `update` function needs to use the argument you pass into it.

Inside the `update` function, change the value of the `button1.innerText` assignment to be `location["button text"]`. That way, you use bracket notation to get the `"button text"` property of the `location` object passed into the function.

Step 65:

`location["button text"]` is an array with three elements. Change the `button1.innerText` assignment to be `location["button text"][0]` which represents the first element of the array.

Step 66:

Now update `button2.innerText` and `button3.innerText` to be assigned the second and third values of the `"button text"` array, respectively.

Step 67:

Following the same pattern as you did for the button text, update the three buttons' `onclick` assignments to be the first, second, and third values of the `"button functions"` array.

Step 68:

Finally, update the `text.innerText` assignment to equal the `text` from the `location` object. However, instead of using bracket notation, use dot notation.

Here is an example of accessing the `name` property of an object called `person`:

Example Code:

```
person.name
```

Step 69:

Now update your `goStore` function to call the `update` function. Pass the second element of the `locations` array as your argument.

To make sure your refactoring is correct, try clicking your first button again. You should see the same changes to your webpage that you saw earlier.

Step 70:

Create two more empty functions named `fightSlime` and `fightBeast`. These functions will be used in your upcoming `cave` object.

Step 71:

Add a third object to the `locations` array. Give it the same properties as the other two objects.

Set `name` to `cave`. Set `button text` to an array with the strings `"Fight slime"`, `"Fight fanged beast"`, and `"Go to town square"`. Set the `"button functions"` to an array with the variables `fightSlime`, `fightBeast`, and `goTown`. Set the `text` property to `"You enter the cave. You see some monsters."`.

Step 72:

Now that you have a `"cave"` location object, update your `goCave` function to call `update` and pass that new `"cave"` location. Remember that this is the third element in your `locations` array.

Don't forget to remove your `console.log` call!

Step 73:

Now that your `"store"` and `"cave"` locations are complete, you can code the actions the player takes at those locations. Inside the `buyHealth` function, set `gold` equal to `gold` minus `10`.

For example, here is how you would set `num` equal to `5` less than `num`: `num = num - 5;`.

Step 74:

After the `gold` is updated, add a line to set `health` equal to `health` plus `10`.

Step 75:

There is a shorthand way to add or subtract from a variable called compound assignment. For example, changing `num = num + 5` to compound assignment would look like `num += 5`.

Update both lines inside your `buyHealth` function to use compound assignment.

Step 76:

Now that you are updating the `gold` and `health` variables, you need to display those new values on the game screen. You have retrieved the `healthText` and `goldText` elements in a prior step.

After your assignment lines, assign the `innerText` property of `goldText` to be the variable `gold`. Use the same pattern to update `healthText` with the `health` variable.

You can test this by clicking your `"Go to store"` button, followed by your `"Buy Health"` button.

Note: Your answer should only be two lines of code.

Step 77:

What if the player doesn't have enough gold to buy health? You should use an `if` statement to check if the player has enough gold to buy health.

In the previous project, you learned how to work with `if` statements like this:

Example Code:

```
const num = 5;

if (num >= 3) {
```

```
  console.log("This code will run because num is greater than or equal to 3.");

}
```

Start by placing all of the code in your `buyHealth` function inside an `if` statement. For the `if` statement condition, check if `gold` is greater than or equal to `10`.

Step 78:

Now when a player tries to buy health, it will only work if they have enough money. If they do not, nothing will happen. Add an `else` statement where you can put code to run if a player does not have enough money.

In the previous project, you learned how to work with `else` statements like this:

Example Code:

```
if (num >= 5) {

} else {

}
```

Step 79:

Inside the `else` statement, set `text.innerText` to equal `"You do not have enough gold to buy health."`

Step 80:

Now the string is no longer printing, because `false` is not `true`. But what about other values?

Try changing the condition to the string "false".

Step 81:

Just like your locations array, your weapons array will hold objects. Add four objects to the weapons array, each with two properties: name and power. The first should have the name set to "stick" and the power set to 5. The second should be "dagger" and 30. The third, "claw hammer" and 50. The fourth, "sword" and 100.

Step 82:

Inside your buyWeapon function, add an if statement to check if gold is greater than or equal to 30.

Step 83:

Similar to your buyHealth function, set gold equal to 30 less than its current value. Make sure this is inside your if statement.

Step 84:

The value of the currentWeaponIndex variable corresponds to an index in the weapons array. The player starts with a "stick", since currentWeaponIndex starts at 0 and weapons[0] is the "stick" weapon.

In the buyWeapon function, use compound assignment to add 1 to currentWeaponIndex - the user is buying the next weapon in the weapons array.

Step 85:

In the previous project, you learned how to use the increment operator to increase a variable by `1`.

Example Code:

```
let num = 5;

num++;

// prints 6

console.log(num);
```

Change your `currentWeaponIndex` assignment to use the increment operator.

Step 86:

Now update the `goldText` element to display the new value of `gold`, and update the `text` element to display `"You now have a new weapon."`.

Step 87:

You should tell the player what weapon they bought. In between the two lines you just wrote, use `let` to initialize a new variable called `newWeapon`. Set this to equal `weapons`.

Step 88:

Use bracket notation to access an object within the `weapons` array and assign it to your `newWeapon` variable. Place the variable `currentWeaponIndex` within the brackets.

When you use a variable in bracket notation, you are accessing the property or index by the *value* of that variable.

For example, this code uses the `index` variable to access a value of `array`.

Example Code:

```
let value = array[index];
```

Step 89:

`weapons[currentWeaponIndex]` is an object. Use dot notation to get the `name` property of that object.

Step 90:

In the previous project, you learned how to work with the concatenation operator to insert variables into a string like this:

Example Code:

```
const organization = "freeCodeCamp";

// "Hello, our name is freeCodeCamp."

"Hello, our name is " + organization + ".";
```

Update the string `"You now have a new weapon."` to `"You now have a "` followed by the name of the new weapon, and remember to end the sentence with a period.

Step 91:

Back at the beginning of this project, you created the `inventory` array. Add the `newWeapon` to the end of the `inventory` array using the `push()` method.

In the previous project, you learned how to work with the `push` method like this:

Example Code:

```
const myArray = [];

myArray.push("new item");

// myArray is now ["new item"]
```

Step 92:

Up until now, any time `text.innerText` was updated, the old text was erased. This time, use the `+=` operator to add text to the end of `text.innerText`.

Add the string `" In your inventory you have: "` - include the spaces at the beginning and the end.

Step 93:

At the end of the second `text.innerText` string you just added, use the concatenation operator to add the contents of `inventory` to the string.

Step 94:

Add an `else` statement to your `buyWeapon` function. In that statement, set `text.innerText` to equal `"You do not have enough gold to buy a weapon."`

Step 95:

Once a player has the best weapon, they cannot buy another one. Wrap all of the code in your `buyWeapon` function inside another `if`

statement. The condition should check if currentWeaponIndex is less than 3 - the index of the last weapon.

Step 96:

Arrays have a length property that returns the number of items in the array. You may want to add new values to the weapons array in the future.

Change your if condition to check if currentWeaponIndex is less than the length of the weapons array. An example of checking the length of an array myArray would look like myArray.length.

Step 97:

Now it is time to test your buyWeapon function. Right now, the gold amount is set to 50. But to properly see the results of your buyWeapon function, the amount should be set to something higher.

Update the gold amount to 250.

NOTE: The HTML has already been updated to reflect this change.

To test your buyWeapon function, open up the console. Then click on the "Go to store" button followed by the "Buy weapon (30 gold)" button four times.

Step 98:

When you were testing your function, you should have seen an error message in the console. This error is due to the condition in the buyWeapon function.

The currentWeaponIndex variable is the index of the weapons array, but array indexing starts at zero. The index of the last element in an array is one less than the length of the array.

Change the `if` condition to check `weapons.length - 1`, instead of `weapons.length`.

Test out your `buyWeapon` function again to see the error message disappear.

Step 99:

If the player has purchased all of the weapons in the inventory, the player should not be able to purchase any more and a message should be displayed.

Add an `else` statement for your outer `if` statement. Inside this new `else` statement, set `text.innerText` to `"You already have the most powerful weapon!"`.

Test your `buyWeapon` function again to make sure the message is displayed when the player has the most powerful weapon.

Step 100:

Now that you are finished testing that portion of the `buyWeapon` function, you can set your `gold` variable back to `50`.

Note: The HTML has already been updated to reflect the original value of `gold`.

Step 101:

Once a player has the most powerful weapon, you can give them the ability to sell their old weapons.

In the outer `else` statement, set `button2.innerText` to `"Sell weapon for 15 gold"`. Also set `button2.onclick` to the function name `sellWeapon`.

Step 102:

Create an empty `sellWeapon` function.

Step 103:

Players should not be able to sell their only weapon. Inside the `sellWeapon` function, add an `if` statement with a condition that checks if the length of the `inventory` array is greater than `1`.

Step 104:

Inside the `if` statement, set `gold` equal to `15` more than its current value. Also update `goldText.innerText` to the new value.

Step 105:

The next step is to create a variable called `currentWeapon`.

Example Code:

```
let num = 1;

if (num === 1) {

  let num = 2; // this num is scoped to the if statement

  console.log(num); // expected output: 2

}

console.log(num); // expected output: 1 (the global variable)
```

Use the `let` keyword to create a variable named `currentWeapon`. Don't assign it a value yet.

Step 106:

In the previous project, you learned how to work with the `shift()` method to remove the first element from an array like this:

Example Code:

```
const myArray = ["first", "second", "third"];

const firstElement = myArray.shift();

// myArray is now ["second", "third"]
```

Use the `shift()` method to take the first element from the `inventory` array and assign it to your `currentWeapon` variable.

Step 107:

After your `currentWeapon`, use the concatenation operator to set `text.innerText` to the string `"You sold a "`, then `currentWeapon`, then the string `"."`.

Step 108:

Now use the `+=` operator to add the string `" In your inventory you have: "` and the contents of `inventory` to the `text.innerText`. Make sure to include the space at the beginning and end of the `" In your inventory you have: "` string.

Step 109:

Use an `else` statement to run when the `inventory` length is not more than one. Set the `text.innerText` to say `"Don't sell your only weapon!"`.

Step 110:

Now you can start the code to fight monsters. To keep your code organized, your `fightDragon` function has been moved for you to be near the other `fight` functions.

Below your `weapons` array, define a `monsters` variable and assign it an array. Set that array to have three objects, each with a `name`, `level`, and `health` properties. The first object's values should be `"slime"`, `2`, and `15`, in order. The second should be `"fanged beast"`, `8`, and `60`. The third should be `"dragon"`, `20`, and `300`.

Step 111:

Fighting each type of monster will use similar logic. Create an empty function called `goFight` to manage this logic.

Step 112:

In your `fightSlime` function, set `fighting` equal to `0` - the index of `slime` in the `monsters` array. Remember that you already declared `fighting` earlier in your code, so you do not need `let` or `const` here.

On the next line, call the `goFight` function.

Step 113:

Following the same pattern as the `fightSlime` function, use that code in the `fightBeast` and `fightDragon` functions. Remember that `beast` is at index `1` and `dragon` is at index `2`. Also, remove the `console.log` call from your `fightDragon` function.

Step 114:

At the end of your code, create two empty functions named `attack` and `dodge`.

Step 115:

Add a new object to the end of the `locations` array, following the same properties as the rest of the objects. Set `name` to `"fight"`, `"button text"` to an array with `"Attack"`, `"Dodge"`, and `"Run"`, `"button functions"` to an array with `attack`, `dodge`, and `goTown`, and `text` to `"You are fighting a monster."`

Step 116:

In the `goFight` function, call your `update` function with the fourth object in `locations` as an argument.

Step 117:

Below your `update` call, set the `monsterHealth` to be the health of the current monster. You can get this value by accessing the `health` property of `monsters[fighting]` with dot notation.

Step 118:

By default, the HTML element that shows the monster's stats has been hidden with CSS. When the player clicks the "Fight dragon" button, the monster's stats should be displayed. You can accomplish this by using the style and display properties on the `monsterStats` element.

The `style` property is used to access the inline style of an element and the `display` property is used to set the visibility of an element.

Here is an example of how to update the display for a paragraph element:

Example Code:

```
const paragraph = document.querySelector('p');

paragraph.style.display = 'block';
```

Display the `monsterStats` element by updating the `display` property of the `style` property to `block`.

Step 119:

Now, you will need to update the text for the current monster's name and health.

Start by assigning `monsters[fighting].name` to the `innerText` property of `monsterName`. Then, assign `monsterHealth` to the `innerText` property of `monsterHealthText`.

Step 120:

Now you can build the `attack` function. First, update the `text` message to say `"The <monster name> attacks."`, replacing `<monster name>` with the name of the monster. Remember you can use the concatenation operator for this.

Step 121:

On a new line, use the addition assignment operator(`+=`), to add the string `" You attack it with your <weapon>."` to the `text` value, replacing `<weapon>` with the player's current weapon. Additionally,

remember that this line of text starts with a space so it will properly display.

Step 122:

Next, set `health` to equal `health` minus the monster's level. Remember you can get this from the `monsters[fighting].level` property.

Step 123:

Set `monsterHealth` to `monsterHealth` minus the power of the player's current weapon.

Remember that you can access the power of the player's current weapon using `weapons[currentWeaponIndex].power`.

Step 124:

The `Math` object in JavaScript contains static properties and methods for mathematical constants and functions. One of those is `Math.random()`, which generates a random number from `0` (inclusive) to `1` (exclusive). Another is `Math.floor()`, which rounds a given number down to the nearest integer.

Using these, you can generate a random number within a range. For example, this generates a random number between `1` and `5`: `Math.floor(Math.random() * 5) + 1;`.

Following this pattern, use the addition operator (`+`) to add a random number between `1` and the value of `xp` to your `monsterHealth -= weapons[currentWeaponIndex].power`.

Step 125:

Update `healthText.innerText` and `monsterHealthText.innerText` to equal `health` and `monsterHealth`.

Step 126:

Add an `if` statement to check if `health` is less than or equal to `0`. If it is, call the `lose` function.

Step 127:

You can make an `else` statement conditional by using `else if`. Here's an example:

Example Code:

```
if (num > 10) {

} else if (num < 5) {

}
```

At the end of your `if` statement, add an `else if` statement to check if `monsterHealth` is less than or equal to `0`. In your `else if`, call the `defeatMonster` function.

Step 128:

At the end of your code, create the `defeatMonster` and `lose` functions. Leave them empty for now.

Step 129:

Inside the dodge function, set text.innerText equal to the string "You dodge the attack from the <monster>". Replace <monster> with the name of the monster, using the name property.

Step 130:

In your defeatMonster function, set gold equal to gold plus the monster's level times 6.7. Remember you can get the monster's level by using monsters[fighting].level.

Here is an example of setting num to num plus 5 * 8: num += 5 * 8. Use Math.floor() to round the result down.

Step 131:

Set xp to xp plus the monster's level.

Step 132:

Now update goldText and xpText to display the updated values.

Step 133:

Finish the defeatMonster function by calling the update function with locations[4] as the argument.

Step 134:

Your locations array doesn't have a fifth element, so locations[4] doesn't work.

Add a new object at the end of the `locations` array, following the same structure as the other objects. Set `name` to `"kill monster"`, set `"button text"` to an array with three `"Go to town square"` strings, set `"button functions"` to an array with three `goTown` variables, and set `text` to `"The monster screams Arg! as it dies. You gain experience points and find gold."`.

Step 135:

The word `"Arg!"` should have quotes around it. Besides escaping quotes, there is another way you can include quotation marks inside a string.

Change the double quotes around the string `"The monster screams Arg! as it dies. You gain experience points and find gold."` to single quotes `'`, then add double quotes around `"Arg!"`.

Step 136:

After a monster is defeated, the monster's stat box should no longer display.

On the first line of the `update` function, use `monsterStats.style.display` to change the `display` value to `none`.

Step 137:

In the `lose` function, call the `update` function and pass in the sixth object of your `locations` array. Note that you haven't created this object just yet.

Step 138:

At the end of your code, create a `restart` function. Inside this function, set `xp` to `0`, `health` to `100`, `gold` to `50`, `currentWeaponIndex` to `0`, and set `inventory` to an array with the string `stick`.

Also update the `innerText` properties of `goldText`, `healthText`, and `xpText` to their current values.

Finally, call the `goTown()` function.

Step 139:

In the `locations` array, add another object at the end. Set the `name` property to `"lose"`, set `"button text"` to an array with three `"REPLAY?"` strings, set `"button functions"` to an array with three `restart` variables, and set `text` to `"You die. &#x2620;"`.

In a later step, you will update the code for the `&#x2620;` emoticon text to properly display on the page.

Step 140:

Back to your `attack` function - inside the `else if` block, create another `if` and `else` statement. If the player is fighting the dragon (`fighting` would be `2`), call the `winGame` function. Move the `defeatMonster()` call to the `else` block.

For this step, you will need to use the strict equality (`===`) operator to check if `fighting` is equal to `2`.

Step 141:

In order for the `&#x2620;` emoticon text to properly display on the page, you will need to use the innerHTML property.

The `innerHTML` property allows you to access or modify the content inside an HTML element using JavaScript.

Here is an example of updating the content for this paragraph element using the innerHTML property.

Example Code:

```
<p id="demo">This is a paragraph.</p>
```

Example Code:

```
document.querySelector("#demo").innerHTML = "Hello, innerHTML!";
```

In the update function, change text.innerText to text.innerHTML.

Step 142:

After the lose function, create a function called winGame. Inside the winGame function, call the update function and pass in locations[6].

Step 143:

Add another object in the locations array. Everything should be the same as the lose object, except the name should be "win" and the text should be "You defeat the dragon! YOU WIN THE GAME! 🎉".

Step 144:

While your game is feature-complete at this stage, there are things you can do to make it more fun and engaging. To get started, you'll give monsters a dynamic attack value.

Inside your attack function, change your health -= monsters[fighting].level; line to health -= getMonsterAttackValue(monsters[fighting].level);. This sets health

equal to `health` minus the return value of the `getMonsterAttackValue` function, and passes the `level` of the monster as an argument.

Step 145:

Below your `attack` function, create an empty function named `getMonsterAttackValue`. It should take `level` as a parameter.

Step 146:

The attack of the monster will be based on the monster's `level` and the player's `xp`. In the `getMonsterAttackValue` function, use `const` to create a variable called `hit`. Assign it the equation `(level * 5) - (Math.floor(Math.random() * xp));`.

This will set the monster's attack to five times their `level` minus a random number between `0` and the player's `xp`.

Step 147:

Log the value of `hit` to the console to use in debugging. Remember that you can do this with `console.log()`.

Step 148:

In the previous project, you learned how to work with the `return` keyword to return a value from a function like this:

Example Code:

```
function add(num1, num2) {

  return num1 + num2;
```

```
}
```

Use the `return` keyword to return the value of `hit` at the end of the function.

Step 149:

If you play the game in its current state you might notice a bug. If your `xp` is high enough, the `getMonsterAttackValue` function will return a negative number, which will actually add to your total health when fighting a monster! You can fix this issue by using a ternary operator to ensure negative values are not returned.

The ternary operator is a conditional operator and can be used as a one-line `if-else` statement. The syntax is: `condition ? expressionIfTrue : expressionIfFalse`.

Here is an example of returning a value using an `if-else` statement and a refactored example using a ternary operator:

Example Code:

```
// if-else statement

if (score > 0) {

  return score

} else {

  return default_score

}

// ternary operator

return score > 0 ? score : default_score
```

In `getMonsterAttackValue`, change `return hit` to a ternary operator that returns `hit` if `hit` is greater than `0`, or returns `0` if it is not.

Step 150:

In your attack function, find the line of code that updates the `monsterHealth` variable and place it within an `if` block with a condition that calls the `isMonsterHit` function.

Step 151:

Add an `else` statement to the first `if` statement inside your `attack()` function. In the `else` statement, use the `+=` operator to add the text `" You miss."` to the end of `text.innerText`.

Step 152:

Now create the `isMonsterHit` function. This will return a boolean value (true or false) to be used in your `if` statement. Return the result of the comparison `Math.random() > .2`.

Step 153:

The player should hit if either `Math.random() > .2` **or** if the player's health is less than `20`.

At the end of your `return` statement, use the logical OR operator `||` and check if `health` is less than `20`.

The logical OR operator will use the first value if it is truthy – that is, anything apart from `NaN`, `null`, `undefined`, `0`, `-0`, `0n`, `""`, and `false`. Otherwise, it will use the second value.

For example: `num < 10 || num > 20`.

Step 154:

On every attack, there should be a chance that the player's weapon breaks. At the end of the `attack` function, add an empty `if` statement with the condition `Math.random() <= .1`.

Step 155:

Use the `+=` operator to add `" Your <weapon> breaks."`, with a space in front of `Your`, to the end of `text.innerText`. Replace `<weapon>` with the last item in the `inventory` array using `inventory.pop()`, which will remove the last item in the array AND return it so it appears in your string.

Step 156:

Remember that the increment operator `++` can be used to increase a variable's value by `1`. There is also a decrement operator `--` that can be used to decrease a variable's value by `1`. For example :

Example Code:

```
let num = 10;

num--;

console.log(num); // Output: 9
```

Decrement the value of `currentWeaponIndex` in your `if` statement, after you update the text.

Step 157:

We don't want a player's only weapon to break. The logical AND operator checks if two statements are true.

Use the logical AND operator `&&` to add a second condition to your `if` statement. The player's weapon should only break if `inventory.length` does not equal (`!==`) one.

Here is an example of an `if` statement with two conditions:

Example Code:

```
if (firstName === "Quincy" && lastName === "Larson") {

}
```

Step 158:

Now you can add a small easter egg (hidden feature) to your game.

Create a new function called `easterEgg` which calls the `update` function with `locations[7]` as the argument.

Step 159:

Create an empty `pick` function with a parameter named `guess`.

Step 160:

Create two new functions named `pickTwo` and `pickEight`.

Inside each of those, call the `pick()` function and pass either `2` or `8` as the argument depending on the function name.

Step 161:

Add another object to your `locations` array. Set `name` to `"easter egg"`, set `"button text"` to an array with the strings `"2"`, `"8"`, and `"Go to town square?"`, set `"button functions"` to an array with the variables `pickTwo`, `pickEight`, and `goTown`, and `text` to `"You find a secret game. Pick a number above. Ten numbers will be randomly chosen between 0 and 10. If the number you choose matches one of the random numbers, you win!"`.

Step 162:

Inside `pick`, use `const` to initialize a variable named `numbers` and set it to an empty array.

Step 163:

After your `numbers` array, create a `while` loop that runs as long as `numbers.length` is less than `10`.

In the previous project, you learned how to work with `while` loops like this:

Example Code:

```
while (condition) {
  // code to run
}
```

Step 164:

Inside your `while` loop, push a random number between `0` and `10` to the end of the `numbers` array. You can create this random number with `Math.floor(Math.random() * 11)`.

Step 165:

After the `while` loop, set `text.innerText` to equal `"You picked <someGuess>. Here are the random numbers:"`. Replace `<someGuess>` with the `guess` function parameter.

Step 166:

At the end of the string, before the final quote, insert the new line escape character `\n`. This will cause the next part you add to `text.innerText` to appear on a new line.

Step 167:

In the previous project, you learned how to work with `for` loops like this:

Example Code:

```
for (let i = 0; i < 5; i++) {

  // code to run

}
```

`for` loops are declared with three expressions separated by semicolons: `for (a; b; c)`, where `a` is the initialization expression, `b` is the condition, and `c` is the final expression.

In this step, create a `for` loop where `i` is initialized to `0`, the loop runs as long as `i` is less than `10`, and `i` is incremented by `1` after each iteration using the increment operator `++`.

Step 168:

Now you can write the logic to run in the loop. Inside your `for` loop, use the `+=` operator to add to the end of `text.innerText`. Add the number at index `i` of the `numbers` array, using `numbers[i]`. Then add a new line, using the escape sequence you used earlier.

Step 169:

The `.includes()` method determines if an array contains an element and will return either `true` or `false`.

Here is an example of the `.includes()` syntax:

Example Code:

```
const numbersArray = [1, 2, 3, 4, 5]

const number = 3



if (numbersArray.includes(number)) {

  console.log("The number is in the array.")

}
```

After your `for` loop, add an `if` statement to check if the `guess` is in the `numbers` array. You can use the `.includes()` method to check if the array contains the `guess`.

Step 170:

Inside the `if` statement, add the string `"Right! You win 20 gold!"` to the end of `text.innerText`. Also, add `20` to the value of `gold` and update the `goldText.innerText`.

Step 171:

Now add an `else` statement. Inside, add `"Wrong! You lose 10 health!"` to the end of `text.innerText`. Subtract `10` from `health` and update `healthText.innerText`.

Step 172:

Since you subtracted health from the player, you need to check if the player's `health` is less than or equal to `0`. If it is, call the `lose` function.

Step 173:

Looking at your `"kill monster"` object, `"button functions"` currently has three `goTown` variables. Replace the third one with `easterEgg` - this is how a player will access the hidden feature of the game. Do not change the `"button text"`.

With this, your RPG game is complete! You can now play around - can you defeat the dragon?