UNIT 03

LESSON 03.02

DOM (Document Object Model) functions

events calling functions (click, change)

Document Object Model (DOM)

JS "sees" the web page as a hierarchical collection of objects, known as the DOM (Document Object Model).

DOM manipultion

JS can make stuff happen and things change on the web page. This is known as "DOM manipultion".

DOM event

A DOM event is something that happens on the web page that can call a function. Functions that manipulate the DOM are typically triggered by a button click, menu change, or other **DOM event**.

HTML elements as JS Objects

Any HTML element can be brought into JS where it becomes an object. The commands for getting elements are:

- **document.getElementById()** gets the element of specified id and stores it in an object
- **document.querySelector()** gets the first element of specified class or tag name and stores it in an object
- **document.querySelectorAll()** gets all elements of specified class or tag name and stores them in an array called a Node List

button.addEventListener()

To enable a button or other element to call a function, attach the **addEventListener** method to the object. The method requires two arguments: an event to 'listen' for and a function to call when the event takes place.

DOM function exploration page

This lesson has a page already set up with the HTML and CSS for several buttons, some input text fields and a select menu.

Open the HTML file in the browser and have a look: lots of buttons and things, but nothing works yet, because it is our job to write the code for that.

return value vs. no return value

Before we get started, it is important to note that these functions do not **return** a value. The function output is being displayed on the web page, rather than being stored in a variable in the script. That said, it is possible to combine return values with DOM output, as we will see.

function #1: greetWorld()

We want the **greet world** button to call a **greetWorld()** function. The button is in the html as: `greet world`

1. In the JS START file, add this function, and call it from the script (no button yet):

```
function greetWorld() {
  alert('Hello World');
}
greetWorld();
```

2. Comment out the function call since a button click is taking over.
3. Get the element `greet world` by id so that we can tell it to call the function. Save it to a variable so that we can tell it to call a function:

```
const btn1 = document.getElementById('btn-1');
```

4. Also get the

   ## output

   tag to hold the output. Save it to a variable so that we can output our function results there:

```
const output = document.getElementById('output');
```

const for objects

As we learned in a previous lesson, **const** for primitive data types (string, number, boolean) cannot be changed in any way.

But **const** works differently for objects:

- **const** objects cannot be *mutated*, which means the *data type* cannot be changed.
- **const** properties, however, *can** be changed.
- **const** protects objects from being accidentally changed to, say, a string, but otherwise you can change/add/remove properties at will.

For these reasons, we typically use **const** to save DOM objects that we bring into JS.

element attributes become object properties

Once in JS, the elements' attributes are the object's properties.

5. Log **btn1** and **object** as well as and some properties:

```
console.log(btn1);
console.log(btn1.id);
console.log(btn1.textContent);
console.log(output);
console.log(output.id);
console.log(output.textContent);
```

addEventListener()

The **addEventListener()** is called on an object and takes two arguments: an event to "listen" for and a function to call when the event takes place.

6. Tell the **btn1** object to "listen" for a click upon itself and to call **greetWorld** when the click occurs.

```
  btn1.addEventListener('click', greetWorld);
```

Notice that it is **greetWorld** *not* **greetWorld()**. This is because the parentheses would instantly call the function. We want the funtion to wait for the click.

7. Run the page and click the button. You should get the Hello World alert.

outputting to the DOM

The function works, so next let's have the output appear on the web page, rather than in a pop up.

8. In the function, comment out the alert and switch to displaying the message as the **textContent** property of the **output** tag:

```
function greetWorld() {
  // alert('Hello World');
  output.textContent = 'Hello World';
}
```

.textContent

The **textContent** property is everything between the tags, but cannot include any html. It is therefore best for simple strings, such as 'Hello World'.

.innerHTML

innerHTML refers to everything between the tags, including html, so tags within tags.

9. Change the output to be a link, which necessitates a switch from *textContent* to *innerHTML*:

```
function greetWorld() {
  // alert('Hello World');
  output.innerHTML = '<a href="#">Hello World</a>';
}
```

function #2: greetByName()

This next function also runs on button click. It gets the user-inputted names from the input fields, concatenates a personalized greeting, and outputs it to the page.

The html is already written:

```
<div>
  <input type="text" id="firstName" placeholder="First Name">
  <input type="text" id="lastName" placeholder="Last Name">
  <button id="btn-2">greet by name</button>
</div>
```

10. Get the button `greet by name`:

```
const btn2 = document.getElementById('btn-2');
```

11. Have the button "listen" for a click and call the function when the click occurs:

```
btn2.addEventListener('click', greetByName);
```

12. Write the function. Get the **values** of the text input fields, which have id's of **firstName** and **lastName**. The values are whatever the user typed:

```
function greetByName() {
    let fName = document.getElementByIc('firstName').value;
    let lName = document.getElementByIc('lastName').value;
    output.textContent = **Hey, ${fname} $lName}**;
}
```

13. Run the page and enter a first and last name.

14. Click the **greet by name** button. The personalized output should appear.

function #3: pickFruit()

This next function runs when the user chooses a fruit from the menu. The html is already there:

```
<select id="fruit-menu">
  <option value="choose">Pick a Fruit:</option>
  <option value="apple">Apple</option>
  <option value="banana">Banana</option>
  <option value="cherry">Cherry</option>
  // etc. more options
</select>
```

The way it works is:

- user chooses a fruit from the **select** menu.
- the choice is a **change** event, which calls the function, which:
  - gets the menu **value**, which equals the value of the selected **option**
  - concatenates and outputs a message to the **output** tag.

15. Get the select menu:

```
let fruitMenu = document.getElementById('fruit-menu');
```

16. Tell it to listen for a **change** event. When it hears the change (to itself), it calls a **pickFruit** function:

```
fruitMenu.addEventListener('change', pickFruit);
```

17. Write the function:

- Get the value of **fruitMenu** and save it to a local variable, **fruit**.
- Append **toUpperCase()** to make the fruit UPPERCASE.
- Concatenate and output a message about the chosen fruit.

```
function pickFruit() {
    let fruit = fruitMenu.value.toUpperCase();
    output.textContent = **Your chosen fruit is: ${fruit}!**;
}
```

function #4: addNumbers()

Next, we will do a function that adds numbers inputted by the user and outputs the sum.

We already have the html for this:

```
<div>
  <input type="number" id="num1-box" placeholder="Enter a Number">
  <input type="number" id="num2-box" placeholder="Enter a Number">
```

```
    <button id="btn-3">Add Numbers</button>
  </div>
```

The **input** elements are set to **type="number"** to prevent non-numeric characters from being entered.

19. Get the **add numbers** button and tell it to call a function when clicked:

```
let btn3 = document.getElementById('btn-3');
btn3.addEventListener('click', addNumbers);
```

20. Write the function. Get the values from the input boxes. Then add the numbers and concatenate the output:

```
function addNumbers() {
  let num1 = document.getElementById('num1-box').value;
  let num2 = document.getElementById('num2-box').value;
  let sum = num1 + num2;
  output.textContent = sum;
}
```

21. Run the page. Enter two numbers and click the **add numbers** button.

There is a bug. The sum is no sum at all, but rather a concatenation of the two numbers. This happened because even though the input boxes are of type number, the "numbers" still came into JS as strings.

22. Convert the numeric input to actual numbers so that the + adds them:

```
  function addNumbers() {
    let num1 = document.getElementById('num1-box').value;
    let num2 = document.getElementById('num2-box').value;
    num1 = Number(num1);
    num2 = Number(num2);
    let sum = num1 + num2;
    output.textContent = sum;
}
```

23. Run the page again. The math should work now.

data-attribute

You can attach additional data to any tag by adding a "data-name" attribute. The data is available in JS as: object.dataset.name

24. Comment out the select menu and "uncomment out" the other select menu--the one where each option has a data-food attribute:

```
    <select id="fruit-menu">
        <option value="choose">Pick a Fruit:</option>
        <option value="apple" data-food="apple sauce">Apple</option>
        <option value="banana" data-food="banana bread">Banana</option>
        - ETC -
    </select>
```

options array

The select object has an options property, which is an array. We will get into arrays in Unit 4, but for now, just know that an array is a variable that holds more than one item at a time. The items are stored by number, called index. We can look up an array item by index. When a menu choice is made, the object selectedIndex property is set with a number that corresponds to the option chosen. We can use that number to look up the selected option from the options array. Once we have that option, can access any **data-** attribute via the **dataset** property. We have an attribute data-food, the value of which is **dataset.food**

Refactor the pickFruit function to include the food:

25. In the function, save the we numeric index of the selected item to a variable. This simplifies the next line.

```
    let indx = fruitMenu.selectedIndex;
```

26. Get the selected option by its index in the options array. Save that to a variable **optn**:

```
    let optn = fruitMenu.options[indx];
```

27. Get the value of the **data-food** attribute for the option:

```
    let food = optn.dataset.food;
```

28. Add the food to the output:

```
      output.textContent = `Your chosen fruit is: ${fruit}! Do you like
  ${food}?`;

    } // end function pickFruit()
```

END LESSON 03.02

NEXT LESSON 03.03