Week 4 Workshop

Web Development Fundamentals

HTML, CSS, and JavaScript

Agenda

Activity	Time
Get Prepared: Log in to Nucamp Learning Portal • Slack • Screenshare	10 minutes
Check-In	10 minutes
Week 4 Recap	60 minutes
Tasks 1 & 2	40 minutes
BREAK	15 minutes
Tasks 2-4	90 minutes
Check-Out	15 minutes

Check-In

- How was this week for you? Any particular challenges or accomplishments?
- Did you understand the Exercises and were you able to complete them?
- How were the Challenges and Quiz this week?
- We know that this was a difficult week for many. Please ask if you have questions.

Week 4 Recap - Overview

New Concepts This Week

- For Loops
- For ... Of Loops
- Break & Continue
- More Array Methods
- More About Strings
  - String Methods
- The HTML DOM
- Locating Nodes
- Creating & Adding Nodes

- Adding Nodes
- Removing Nodes
- Cloning Nodes
- Inline OnEvent Handlers
- Mouse Events
- setTimeout() & setInterval()
- clearTimeout & clearInterval()
- addEventListener()
- removeEventListener()

Next slides will review these concepts

For Loops

- For loops have built-in support to set a variable to iterate for the loop condition:

```
let x = 5;
for (let i = 1; i <= 3; i++) {
    x = x * i;
    console.log(x);
}
```

Discuss: What would the above loop log to the console? Do you understand why?
Type it into your console to try it out.

Note: the statement `x = x * i;` could also be written more simply as: `x *= i;`

For … Of Loops

- for … of loops can be used with arrays and other iterables in JavaScript.

```
const exampleArray = [2, 4, 6, 8];
for (const n of exampleArray) {
    console.log(n * 2);
}
```

Discuss: What does the above log to the console?

Type it into your console to try it out.

Break & Continue

- Use in any for or while loop to break out of the loop entirely (break), or skip the current iteration (continue).
- Given the code to the right, what is the response if you enter a guess of –1 when prompted?
- What about a guess of true?

```
let fingers = Math.floor(Math.random() * 10) + 1;
let guess = 0;
while (guess !== fingers) {
    guess = +prompt('How many fingers am I holding up?');
    if (isNaN(guess)) {
        break;
    }
    if ((guess < 0) || (guess > 10)) {
        continue;
    }
    if (guess === fingers) {
        alert('You got it!');
    } else {
        alert('Try again!');
    }
}
```

More Array Methods

- concat() - combine two arrays into one
- sort() - sort an array of strings alphabetically
  - Note: When using numbers with sort(), results may not be what you expect. [1, 13, 1000, 29, 255].sort() would result in this array:
    - [1, 1000, 13, 255, 29]
    - NOT [1, 13, 29, 255, 1000] as you might expect
- reverse() reverses the order of an array

More Array Methods (cont)

- slice() - copy a part of an array and return a new array – this does not mutate the original array
  splice() - inserts, replaces, or removes parts of an array – this DOES mutate the original array!

- If you run this code:

```
const birds = ['crow', 'hawk', 'owl', 'goose'];
let newBirds = birds.slice(1, 3);
```

- The variable newBirds now holds the array: ['hawk', 'owl']
  The array in the variable birds remains the same.

- If you then ran the code:

```
newBirds = birds.splice(1, 3);
```

- The variable newBirds now holds the array: [hawk, 'owl', 'goose']
  The array in the variable birds is now only: ['crow']

Slice and splice look very similar and do somewhat similar things, but they work very differently so be careful with them.

More About Strings

- Strings are *not* arrays but in some ways work similarly to arrays
- You can access each character in a string by its index, as if each character is an item in an array –
  - `'hello'[0]` is `'h'`
  - `'hello'[1]` is `'e'5`
  - Etc
- And you can get the length of a string with .length:
  - `'hello'.length` is `5`
- **Discuss:** Given this variable:

```
let myStr = 'dog';
```

Can you change `myStr`'s value from **`'dog'`** to **`'zog'`** using the code as follows?:

```
myStr[0] = 'z';
```

(answer in next slide)

String Methods

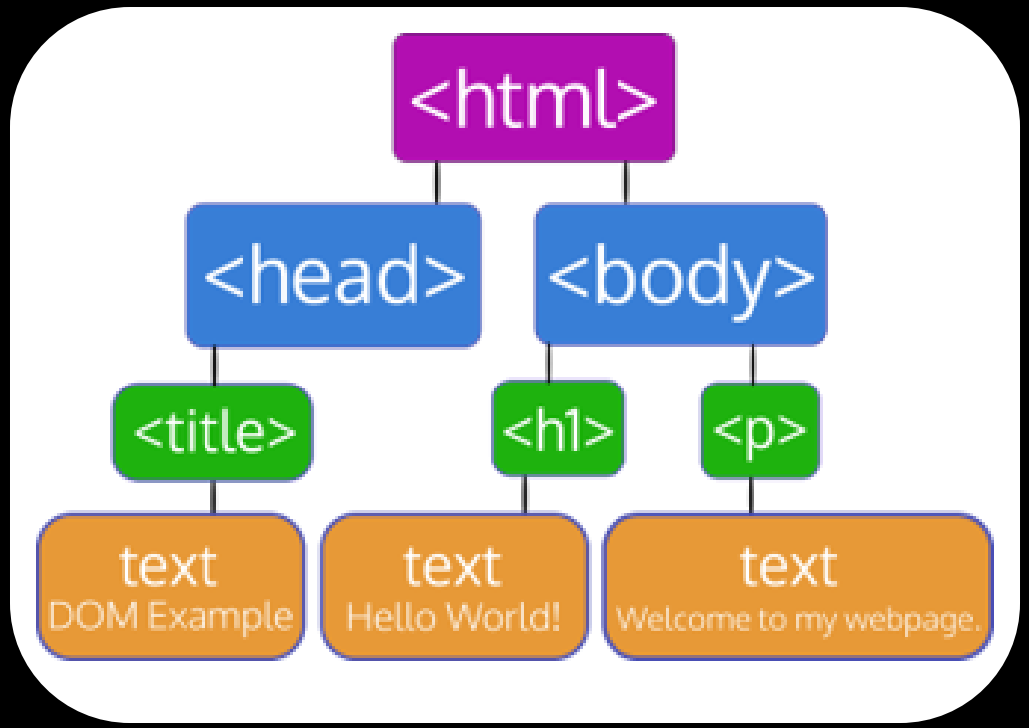
- **Answer:** No! Access to a string's characters via bracket notation is read only. You cannot use that syntax to change the string like you can with arrays.
- Strings are iterables! That means for … of loops work with strings.
- Try this in your console:

```
for (const char of 'foobar') {
    console.log(char);
}
```

  - Strings also have their own methods, including some they share with arrays, such as indexOf(), includes(). slice()
  - **Discuss**: Can anyone name any string methods that are not shared with arrays?

HTML DOM & Node Relationships

- Let's review:



In this example DOM tree...

- What node is the parentNode of <head>?
- What node is the nextSibling of <h1>?
- What node is the previousSibling of <p>?
- What node is the firstChild of <body>?
- What node is the lastChild of <html>?

More Node Relationships

- **Discuss**: What is the difference between `firstChild` and `firstElementChild`, and `lastChild` and `lastElementChild`?

Locating Nodes

- Given this HTML:

```
<html>
<body>
   <h1 id="myH1">Find Me</h1>
</body>
</html>
```

- **Discuss:** Which of the following ways will correctly find this node?
  - A) document.getElementsByTagName('h1');
  - B) document.getElementById('myH1')
  - C) document.querySelector('myH1')
- (answer in next slide)

Locating Nodes

- Answer:
  - A) document.getElementsByTagName('h1');
  - B) document.getElementById('myH1')
  - C) document.querySelector('myH1')
- Only **B** will work! Why?
- A) will not work because getElementsByTagName will return an array, and you need to give it an index for the array item. This would have worked:
  - document.getElementsByTagName('h1')[0]
- C) will not work because querySelector expects its argument to be in the form of a CSS selector. So this would have worked (you need the **#**):
  - document.querySelector('#myH1')

Creating and Adding Nodes

- To create a new element node in JavaScript, use **document.createElement('elementname')**, e.g. document.createElement('div')
- To create a new text node, use **document.createTextNode('your text')**
- Once you have created a new node, it isn't part of the DOM yet, you must add it to the DOM using a node method such as *node*.appendChild() or *node*.insertBefore()

Removing Nodes

- Two ways to remove nodes:
  - *node*.removeChild(*child node to remove*)
  - *node*.remove()
- Example – let's say you have this HTML:

```
<p id="myP"><img src="cat.jpg" /></p>
```

- And this JavaScript:

```
const myPara = document.querySelector('#myP');
```

- If you did this:
  - `myPara.remove()` // This would remove the entire **<p>** element and its child **<img>**.
- If you did this:
  - `myPara.removeChild(myPara.firstChild);` // This would remove just the **<img>** element.

Removing Nodes

- Use a while loop to remove all the child nodes of a parent node without removing the parent node:

```
while (node.firstChild) {
    node.removeChild(node.firstChild);
}
```

Discuss: Any volunteers who would like to try to walk through and explain the above code in their own words?

Cloning Nodes

- **Discuss**: What is the difference between using *node*.cloneNode() and *node*.cloneNode(true)?

Inline OnEvent Handlers

- There are various events in JavaScript such as click, mouseover, etc
- You can set functions to respond to these events, these are called event handler functions, or just event handlers
- To set an inline event handler in HTML, you use the name of the event such as 'click', add 'on' to it – so 'onclick', 'onmouseover', etc
- Then you use it as an attribute in an HTML element:

```
<div onclick="runThisFunction();">Click Me</div>
```

- Give the function to run as the attribute value, use quotes around it and include the argument list, e.g. "runThisFunction()"

Mouse Events

- You learned these inline on-event handlers for mouse events:
  - onclick
  - onmousedown
  - onmouseup
  - onmouseover
  - onmouseout
  - Discuss: What's the difference between **onmousedown** and **onclick**?

setTimeout() & setInterval()

- Set timer events in JavaScript to delay the execution of a function for a given length of time
- setTimeout calls a function once after the time has run out
- setInterval calls a function repeatedly, resetting the clock once the time runs out
- You call each the same way: the first argument is a function name, the second is a time in milliseconds:
  - **setTimeout(doSomething, 10000);** // will call doSomething() in 10 seconds
  - **setInterval(doSomething, 10000);** // will call doSomething(() every 10 seconds

ClearTimeout & clearInterval

- setTimeout() and setInterval() both return a unique timer ID as their return value when you call them
- Save these return values to a variable so that you can use them to clear the timer if necessary. Otherwise you have no way to stop the timer if you need to. E.g.:
- let timerID = setInterval(doSomething, 10000);
- clearInterval(timerID); // stop the interval timer

addEventListener

- addEventListener() is the recommended way in vanilla JS to add event handler functions to events
- Use it like this:
  - node.addEventListener('eventname', functionname);
- Example: You want to have a button respond to clicks by calling a function named doSomething. You have the button node in a variable called btnNode already. You can do this:

```
btnNode.addEventListener('click', doSomething);
```

  - Note: no **'on'** prefix, it's 'click' and not 'onclick'
  - The event name goes inside quotes, single or double
  - The function name does NOT go inside quotes, and you don't include the argument list – it's just doSomething, not 'doSomething' or doSomething()

addEventListener() (cont)

- btnNode.addEventListener('click', doSomething);
  - Note: no 'on' prefix, it's 'click' and not 'onclick'
  - The event name goes inside quotes, single or double
  - The function name does NOT go inside quotes, and you don't include the argument list – it's just doSomething, not 'doSomething' or doSomething()

- Remember: addEventListener can be used with the same event, on the same node, more than once. Other ways of attaching event handlers (such as inline onevent handlers like onclick) do not allow for this.

removeEventListener()

- Use just like addEventListener() to remove an event handler function from an event
- So if you used this to add an event handler function to an event:
- btnNode.addEventListener('click', doSomething);
- You can use this to remove it:
- btnNode.removeEventListener('click', doSomething);

If there is time...

- ...left from what the Agenda slide allocated for the Recap section, then this is the time to bring up any unresolved questions, and to discuss any Challenge Questions or Code Challenges.
- Otherwise, please continue to the Workshop Assignment and save the discussion for after the assignment is finished, or online.

Workshop Assignment

- It's time to start the workshop assignment!
- Break out into groups of 2-3. Sit near your workshop partner(s).
  - Your instructor may assign partners, or have you choose.
- Work closely with each other.
  - Don't forget that the 20-minute rule becomes the 10-minute rule during workshops!
  - 10-minute rule does *not* apply to talking to your partner(s). Work together throughout. This will be useful practice for working with teams in real life.
- Follow the workshop instructions very closely.
  - Talk to your instructor if any of the instructions are unclear to you.

Assignment Submission & Check-Out

- Submit the matching-game.html page at the bottom of the assignment page in the learning portal.