JavaScript

Scripting Language and Basics of JavaScript

- Document Object Model (DOM)
  - Document Object Model (DOM) - Part 1
    - Getting Element
      - Getting elements by tag name
      - Getting elements by class name
      - Getting an element by id
      - Getting elements by using querySelector methods
    - Adding attribute
      - Adding attribute using setAttribute
      - Adding attribute without setAttribute
      - Adding class using classList
      - Removing class using remove
    - Adding Text to HTML element
      - Adding Text content using textContent
      - Adding Text Content using innerHTML
        - Text Content
        - Inner HTML
    - Adding style
      - Adding Style Color
      - Adding Style Background Color
      - Adding Style Font Size
  - DOM(Document Object Model)-Part 2
    - Creating an Element
    - Creating elements
    - Appending child to a parent element
    - Removing a child element from a parent node
  - DOM(Document Object Model)-Part 3
    - Event Listeners
      - Click
      - Double Click
      - Mouse enter
    - Getting value from an input element
    - input value
      - input event and change
      - blur event
      - keypress, keydow and keyup

Document Object Model (DOM)

Document Object Model (DOM) - Part 1

HTML document is structured as a JavaScript Object. Every HTML element has a different properties which can help to manipulate it. It is possible to get, create, append or remove HTML elements using JavaScript. Check the examples below. Selecting HTML element using JavaScript is similar to selecting using CSS. To select an HTML element, we use tag name, id, class name or other attributes.

Getting Element

We can access already created element or elements using JavaScript. To access or get elements we use different methods. The code below has four *h1* elements. Let us see the different methods to access the *h1* elements.

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <title>Document Object Model</title>
    </head>
    <body>

     <h1 class='title' id='first-title'>First Title</h1>
     <h1 class='title' id='second-title'>Second Title</h1>
     <h1 class='title' id='third-title'>Third Title</h1>
     <h1></h1>

    </body>
  </html>
```

Getting elements by tag name

getElementsByTagName():takes a tag name as a string parameter and this method returns an HTMLCollection object. An HTMLCollection is an array like object of HTML elements. The length property provides the size of the collection. Whenever we use this method we access the individual elements using index or after loop through each individual items. An HTMLCollection does not support all array methods therefore we should use regular for loop instead of forEach.

```
// syntax
document.getElementsByTagName('tagname')
```

```
const allTitles = document.getElementsByTagName('h1')

console.log(allTitles) //HTMLCollections
console.log(allTitles.length) // 4

for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i]) // prints each elements in the HTMLCollection
}
```

Getting elements by class name

getElementsByClassName() method returns an HTMLCollection object. An HTMLCollection is an array like list of HTML elements. The length property provides the size of the collection. It is possible to loop through all the HTMLCollection elements. See the example below

```
//syntax
document.getElementsByClassName('classname')
```

```
const allTitles = document.getElementsByClassName('title')

console.log(allTitles) //HTMLCollections
console.log(allTitles.length) // 4

for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i]) // prints each elements in the HTMLCollection
}
```

Getting an element by id

getElementById() targets a single HTML element. We pass the id without # as an argument.

```
//syntax
document.getElementById('id')
```

```
let firstTitle = document.getElementById('first-title')
console.log(firstTitle) // <h1>First Title</h1>
```

Getting elements by using querySelector methods

The *document.querySelector* method can select an HTML or HTML elements by tag name, by id or by class name.

querySelector: can be used to select HTML element by its tag name, id or class. If the tag name is used it selects only the first element.

```
let firstTitle = document.querySelector('h1') // select the first available h1
element
let firstTitle = document.querySelector('#first-title') // select id with first-
title
let firstTitle = document.querySelector('.title') // select the first available
element with class title
```

querySelectorAll: can be used to select html elements by its tag name or class. It returns a nodeList which is an array like object which supports array methods. We can use ***for loop*** or ***forEach*** to loop through each nodeList elements.

```
const allTitles = document.querySelectorAll('h1') # selects all the available h1
elements in the page

console.log(allTitles.length) // 4
for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i])
}

allTitles.forEach(title => console.log(title))
const allTitles = document.querySelectorAll('.title') // the same goes for
selecting using class
```

Adding attribute

An attribute is added in the opening tag of HTML which gives additional information about the element. Common HTML attributes: id, class, src, style, href,disabled, title, alt. Lets add id and class for the fourth title.

```
const titles = document.querySelectorAll('h1')
titles[3].className = 'title'
titles[3].id = 'fourth-title'
```

Adding attribute using setAttribute

The ***setAttribute()*** method set any html attribute. It takes two parameters the type of the attribute and the name of the attribute. Let's add class and id attribute for the fourth title.

```
const titles = document.querySelectorAll('h1')
titles[3].setAttribute('class', 'title')
titles[3].setAttribute('id', 'fourth-title')
```

Adding attribute without setAttribute

We can use normal object setting method to set an attribute but this can not work for all elements. Some attributes are DOM object property and they can be set directly. For instance id and class

```
//another way to setting an attribute
titles[3].className = 'title'
titles[3].id = 'fourth-title'
```

Adding class using classList

The class list method is a good method to append additional class. It does not override the original class if a class exists rather it adds additional class for the element.

```
//another way to setting an attribute: append the class, doesn't over ride
titles[3].classList.add('title', 'header-title')
```

Removing class using remove

Similar to adding we can also remove class from an element. We can remove a specific class from an element.

```
//another way to setting an attribute: append the class, doesn't over ride
titles[3].classList.remove('title', 'header-title')
```

Adding Text to HTML element

An HTML is a build block of an opening tag, a closing tag and a text content. We can add a text content using the property *textContent* or *innerHTML.

Adding Text content using textContent

The *textContent* property is used to add text to an HTML element.

```
const titles = document.querySelectorAll('h1')
titles[3].textContent = 'Fourth Title'
```

Adding Text Content using innerHTML

Most people get confused between *textContent* and *innerHTML*. *textContent* is meant to add text to an HTML element, however innerHTML can add a text or HTML element or elements as a child.

Text Content

We assign *textContent* HTML object property to a text

```
const titles = document.querySelectorAll('h1')
titles[3].textContent = 'Fourth Title'
```

Inner HTML

We use innerHTML property when we like to replace or a completely new children content to a parent element. It value we assign is going to be a string of HTML elements.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript for Everyone:DOM</title>
  </head>
  <body>
    <div class="wrapper">
        <h1>Computer Engineering challenges in 2020</h1>
        <h2>JavaScript Challenge</h2>
        <ul></ul>
    </div>
    <script>
    const lists = `
      <li>Coffee</li>
      <li>Tea</li>
      <li>Milk</li>`
  const ul = document.querySelector('ul')
  ul.innerHTML = lists
    </script>
  </body>
</html>
```

The innerHTML property can allow us also to remove all the children of a parent element. Instead of using removeChild() I would recommend the following method.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript for Everyone:DOM</title>
  </head>
  <body>
    <div class="wrapper">
        <h1>SVNIT</h1>
        <h2>Computer Engineering</h2>
        <ul>
          <li>Coffee</li>
          <li>Tea</li>
          <li>Milk</li>
        </ul>
    </div>
    <script>
  const ul = document.querySelector('ul')
  ul.innerHTML = ''
    </script>
  </body>
</html>
```

Adding style

Adding Style Color

Let us add some style to our titles. If the element has even index we give it green color else red.

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px font size
  if (i % 2 === 0) {
    title.style.color = 'green'
  } else {
    title.style.color = 'red'
  }
})
```

Adding Style Background Color

Let us add some style to our titles. If the element has even index we give it green color else red.

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px font size
  if (i % 2 === 0) {
    title.style.backgroundColor = 'green'
  } else {
    title.style.backgroundColor = 'red'
  }
})
```

Adding Style Font Size

Let us add some style to our titles. If the element has even index we give it 20px else 30px

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px font size
  if (i % 2 === 0) {
    title.style.fontSize = '20px'
  } else {
    title.style.fontSize = '30px'
  }
})
```

As you have notice, the properties of css when we use it in JavaScript is going to be a camelCase. The following CSS properties change from background-color to backgroundColor, font-size to fontSize, font-

family to fontFamily, margin-bottom to marginBottom.

DOM(Document Object Model)-Part 2

Creating an Element

To create an HTML element we use tag name. Creating an HTML element using JavaScript is very simple and straight forward. We use the method *document.createElement()*. The method takes an HTML element tag name as a string parameter.

```
// syntax
document.createElement('tagname')
```

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>

    <script>
        let title = document.createElement('h1')
        title.className = 'title'
        title.style.fontSize = '24px'
        title.textContent = 'Creating HTML element DOM Day 2'

        console.log(title)
    </script>
</body>

</html>
```

Creating elements

To create multiple elements we should use loop. Using loop we can create as many HTML elements as we want. After we create the element we can assign value to the different properties of the HTML object.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>
```

```
<body>

    <script>
        let title
        for (let i = 0; i < 3; i++) {
            title = document.createElement('h1')
            title.className = 'title'
            title.style.fontSize = '24px'
            title.textContent = i
            console.log(title)
        }
    </script>
</body>

</html>
```

Appending child to a parent element

To see a created element on the HTML document we should append it to the parent as a child element. We can access the HTML document body using *document.body*. The *document.body* support the *appendChild()* method. See the example below.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>

    <script>
        // creating multiple elements and appending to parent element
        let title
        for (let i = 0; i < 3; i++) {
            title = document.createElement('h1')
            title.className = 'title'
            title.style.fontSize = '24px'
            title.textContent = i
            document.body.appendChild(title)
        }
    </script>
</body>
</html>
```

Removing a child element from a parent node

After creating an HTML, we may want to remove element or elements and we can use the *removeChild()* method.

Example:

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Removing child Node</h1>
    <h2>SVNIT</h1>
    <ul>
      <li>Coffee</li>
      <li>Tea</li>
      <li>Milk</li>
    </ul>

    <script>
        const ul = document.querySelector('ul')
        const lists = document.querySelectorAll('li')
        for (const list of lists) {
            ul.removeChild(list)

        }
    </script>
</body>

</html>
```

As we have see in the previous section there is a better way to eliminate all the inner HTML elements or the children of a parent element using the method *innerHTML* properties.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Removing child Node</h1>
    <h2>SVNIT</h1>
    <ul>
      <li>Coffee</li>
      <li>Tea</li>
      <li>Milk</li>
    </ul>

    <script>
```

```
        const ul = document.querySelector('ul')
        ul.innerHTML = ''
    </script>
</body>

</html>
```

The above snippet of code cleared all the child elements.

DOM(Document Object Model)-Part 3

Event Listeners

Common HTML events:onclick, onchange, onmouseover, onmouseout, onkeydown, onkeyup, onload. We can add event listener method to any DOM object. We use ***addEventListener()*** method to listen different event types on HTML elements. The *addEventListener()* method takes two arguments, an event listener and a callback function.

```
selectedElement.addEventListener('eventlistner', function(e) {
  // the activity you want to occur after the event will be in here
})
// or

selectedElement.addEventListener('eventlistner', e => {
  // the activity you want to occur after the event will be in here
})
```

Click

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event.

Example: click

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button>Click Me</button>

    <script>
      const button = document.querySelector('button')
```

```
      button.addEventListener('click', e => {
        console.log('e gives the event listener object:', e)
        console.log('e.target gives the selected element: ', e.target)
        console.log(
          'e.target.textContent gives content of selected element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
</html>
```

An event can be also attached directly to the HTML element as inline script.

Example: onclick

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button onclick="clickMe()">Click Me</button>
    <script>
      const clickMe = () => {
        alert('We can attach event on HTML element')
      }
    </script>
  </body>
</html>
```

Double Click

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event. **Example: dblclick**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button>Click Me</button>
    <script>
      const button = document.querySelector('button')
```

```
      button.addEventListener('dblclick', e => {
        console.log('e gives the event listener object:', e)
        console.log('e.target gives the selected element: ', e.target)
        console.log(
          'e.target.textContent gives content of selected element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
</html>
```

Mouse enter

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event.

Example: mouseenter

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button>Click Me</button>
    <script>
      const button = document.querySelector('button')
      button.addEventListener('mouseenter', e => {
        console.log('e gives the event listener object:', e)
        console.log('e.target gives the selected element: ', e.target)
        console.log(
          'e.target.textContent gives content of selected element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
</html>
```

By now you are familiar with addEventListen method and how to attach event listener. There are many types of event listeners. But in this challenge we will focus the most common important events. List of events:

- click - when the element clicked
- dblclick - when the element double clicked
- mouseenter - when the mouse point enter to the element

- mouseleave - when the mouse pointer leave the element
- mousemove - when the mouse pointer move on the element
- mouseover - when the mouse pointer move on the element
- mouseout -when the mouse pointer out from the element
- input -when value enter to input field
- change -when value change on input field
- blur -when the element is not focused
- keydown - when a key is down
- keyup - when a key is up
- keypress - when we press any key
- onload - when the browser has finished loading a page

Test the above event types by replacing event type in the above snippet code.

Getting value from an input element

We usually fill forms and forms accept data. Form fields are created using input HTML element. Let us build a small application which allow us to calculate body mas index of a person using two input fields, one button and one p tag.

input value

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

  <body>
    <h1>Body Mass Index Calculator</h1>

    <input type="text" id="mass" placeholder="Mass in Kilogram" />
    <input type="text" id="height" placeholder="Height in meters" />
    <button>Calculate BMI</button>

    <script>
      const mass = document.querySelector('#mass')
      const height = document.querySelector('#height')
      const button = document.querySelector('button')

      let bmi
      button.addEventListener('click', () => {
        bmi = mass.value / height.value ** 2
        alert(`your bmi is ${bmi.toFixed(2)}`)
        console.log(bmi)
      })
    </script>
  </body>
</html>
```

input event and change

In the above example, we managed to get input values from two input fields by clicking button. How about if we want to get value without click the button. We can use the *change* or *input* event type to get data right away from the input field when the field is on focus. Let us see how we will handle that.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

  <body>
    <h1>Data Binding using input or change event</h1>

    <input type="text" placeholder="say something" />
    <p></p>

    <script>
      const input = document.querySelector('input')
      const p = document.querySelector('p')

      input.addEventListener('input', e => {
        p.textContent = e.target.value
      })
    </script>
  </body>
</html>
```

blur event

In contrast to *input* or *change*, the *blur* event occur when the input field is not on focus.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Giving feedback using blur event</h1>

    <input type="text" id="mass" placeholder="say something" />
    <p></p>

    <script>
        const input = document.querySelector('input')
        const p = document.querySelector('p')
```

```
        input.addEventListener('blur', (e) => {
            p.textContent = 'Field is required'
            p.style.color = 'red'

        })
    </script>
</body>

</html>
```

keypress, keydow and keyup

We can access all the key numbers of the keyboard using different event listener types. Let us use keypress and get the keyCode of each keyboard keys.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

  <body>
    <h1>Key events: Press any key</h1>

    <script>
      document.body.addEventListener('keypress', e => {
        alert(e.keyCode)
      })
    </script>
  </body>
</html>
```
