JavaScript-Essentials 2

DOM Object
Data-Time
Local storage
Time Intervals

What is a Dom Object?

- Dom stands for Document Object Model
- Dom is created by browser and is the interface between the browser and the web page
- programmatically it allows us to interact with the web page
- Things we can do with dom
  - Change, remove and edit HTML elements
  - change and remove and add CSS styles
  - read and change element attributes
  - add event listeners like onclick, onmouseover, onmouseout, keypress, etc

Getting element by ID

- grab some element to do anything! **querying the dom**
- `document` object has all the elements in the page, it is the root of the dom tree
- when you need to select one element, you use the `getElementById` method and pass in the id of the element you want to select `getElementById(id)`
- we can also store this in `var` to acess them later

```
var myElement = document.getElementById('myElement');
```

Getting elements class name or Tag name

- when we want several elemetns we query all classes
- `getElementsByClassName('myClassName')` and `getElementsByTagName(p)`
- lets say i want to get all `li` tags in my document and store it in varibale I will use `var myLi = document.getElementsByTagName('li')`

looping through elements

- we can loop through all the elements in the dom with `forEach`
- for example, if we want to change the color of all the `li` tags in the document, we can do it like this:

```
var myLi = document.getElementsByTagName('li');
myLi.forEach(function(li){
  li.style.color = 'red';
});
```

“

- check if an element is an array using `Array.isArray(myLi)`
- change an html collection to array using `Array.from(myHTMLCollection)`

”

Query Selector

- `querySelector` is a method that allows us to select one element from the dom
- `querySelectorAll` is a method that allows us to select all elements from the dom
  example:

```
// we grab second list element indsiide a p tag which is inside a section with ID #myElement
var myElement = document.querySelector('#myElement p ul li:nth-child(2)');
// get all li elements inside a section with ID #myElement
var allMyElements = document.querySelectorAll('#myElement p ul li');
```

Editing and adding text in DOM

- we can edit text in the dom with `innerHTML` will render html tags!
- wen can also add text with `innerText`
- to get the text content of queried element we use `textContent` wont render html tags
- we can now change the or add the textContent

example 1:

```
var myElement = document.querySelector('#myElement p ul li:nth-child(2)');
myElement.textContent = 'new text';
```

node in DOM

- everything in DOM is a node
- there are multiple node types:
  - text node, element node, comment node, document node, document fragment node
- we can see the type of node using `nodeType`
- we can also get the node name using `nodeName`
- we can find if a node has a child node using `hasChildNodes()`
- clone a node using `cloneNode(true)`, true to include everything in the node
  example:

Traversing parent and child element

- we can traverse the dom with `parentNode` or `parentElement` and `childNodes`
- we can chain these methods to get to the parent node or parent element example below:
- we get line breaks as well with `childNodes` we can igonre them with `children`
- example:

```
var myElement = document.querySelector('#myElement p ul li:nth-child(2)');
console.log(myElement.parentNode.nodeName); // gives us the name of the parent node
```

Traversing same level elements i.e siblings

- we can traverse the dom with `nextSibling` and `previousSibling`
- exmaple of Traversing siblings

```
var myElement = document.querySelector('#myElement p ul li:nth-child(2)');
var myNextSibling = myElement.nextSibling;
var myPreviousSibling = myElement.previousSibling;
```

callback functions

- we can pass a function as an argument to another function
- this function is invoked when the event happens

DOM events

- we can add event listeners to elements using `addEventListener`
- example of some events are click event, mouseover event, mouseout event, keypress event, etc

```
var deleteButton = document.querySelectorAll('.delete');
deleteButton.forEach(function(element, index) {
  element.addEventListener('click', function(event) {
    alert(`You clicked delete button ${index}`);
    element.parentNode.remove();
  })
});
```

prevent Default behavior

- prevent the browser from doing its default behavior when a link is clicked or a form submitted
- we used `e.preventDefault()` we are now preventing the default behavior of the element click action

Event bubbling

- when an event happens, it is first sent to the element that triggered the event
- if the event is not handled by the parent element, it is sent to the parent of the parent element and so on
- at the end of the event chain, the event is sent to the window object
- this is called event bubbling
- ideally we want to attach event listeners to the parent element and not the child element the reason for doing this is that if we attach event listeners to the child element, the event will be triggered twice, once for the child element and once for the parent element

Event lister on parent

- What is more elemetns were added in future the event listener will not work for them for that reason we attach the event listener to the parent element
- We will see if the clicked button is a delete button by checking the class name

```
var deleteButton = document.querySelector('.recommendationList');
deleteButton.addEventListener('click', function(event) {
  if (event.target.className === 'delete') {
    event.target.parentNode.remove();
  }
});
```

Interaction with forms

- `document.forms` will list an html collection of all the forms in the dom
- you can get the form by array index or the form id eg: `document.forms[0]` or `document.forms.myForm`
- on click form emits first the `submit` event and then the `reset` event
- we will listen for submit event and then prevent the default behavior of the form using `e.preventDefault()`

```
var myForm = document.forms.myForm;
myForm.addEventListener('submit', function(event) {
  event.preventDefault();
  console.log('you submitted the form');
});
```

creating elements/adding to dom

- we can create elements using `document.createElement`
- add them to the dom using methods like `appendChild` or `insertBefore`

```
var newElement = document.createElement('li');
newElement.textContent = 'new element';
var myElement = document.querySelector('#myElement p ul');
myElement.appendChild(newElement);
// inser before an element
document.querySelector('#recommendationList ul').insertBefore(newRecommendationElement,
document.querySelector('#recommendationList ul li'));
```

add att and class to elements

- get attribute of a li element `li.getAttribute('class')`
  set `setAttribute`, remove using `removeAttribute`, has att using `hasAttribute`
- set class name using `className` or add/remove to existing `classList.add` or `classList.remove`
- add styles using `style` property of the using `style.cssText = 'color: red; margin: 20px;'`

```
var myElement = document.querySelector('#myElement p ul li:last-child');
myElement.setAttribute('title', 'my title');
myElement.classList.add('myClass');
```

change event

- we can listen for change event using `addEventListener` for check box change we usee `change` event
- change event happens when the value of the element changes
- example:

```
var myCheckbox = document.querySelector('#myCheckbox');
myCheckbox.addEventListener('change', function(event) {
  console.log('you changed the checkbox');
});
```

search filter

- we can use the `value` property of the input element to get the value of the input
- we can use the key up event to listen for the user to type in the search box
- we will grab the text in li tag we will only show those that match the search text

```
var myInput = document.querySelector('#myInput');
var myList = document.querySelector('#myList');
myInput.addEventListener('keyup', function(event) {
  var searchText = event.target.value;
  var listItems = myList.querySelectorAll('li');
  listItems.forEach(function(element, index) {
    if (element.textContent.indexOf(searchText) !== -1) {
      element.style.display = 'block';
    } else {
      element.style.display = 'none';
    }
  });
});
```

Tabbed content

- add tag and show realated content to that tab.
- when we click and li tag we will look for data-target attr and show the content that is related to that data-target attr by setting the class to active

```
var myTab = document.querySelector('#myTab');
myTab.addEventListener('click', function(event) {
  if (event.target.tagName === 'LI') {
    var myTabContent = document.querySelector('#myTabContent');
    var myTarget = event.target.dataset.target;
    var myTargetElement = myTabContent.querySelector('.' + myTarget);
    var myTabs = myTabContent.querySelectorAll('li');
    myTabs.forEach(function(element, index) {
      element.classList.remove('active');
    });
    myTargetElement.classList.add('active');
  }
});
```

final DOM! DomContentLoadedEvent

- we can attch event when the script is in head
- sometime we want to add the file on head so we use `DomContentLoadedEvent` for example

```
document.addEventListener('DOMContentLoaded', function(event) {
  console.log('DOM is ready');
});
```

The date Object

- create a date object using `new Date()` OP:
  `Sun May 20 2018 00:00:00 GMT+0200 (Eastern European Summer Time)`
- you can pass values to the data object like year month day past date `new Date(1995, 11, 17)` //1995, 11, 17
- future date `new Date(2025, 11, 17)` //2025, 11, 17
- retrieve date realted info `getDate()` `getDay()` `getFullYear()` `getHours()` `getMinutes()` `getSeconds()` `getMilliseconds()`
- set date realted info `setDate()` `setFullYear()` `setHours()` `setMinutes()` `setSeconds()` `setMilliseconds()`
- Date takes in argument like

```
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

the getTime() method

- the getTime() method returns the number of milliseconds since January 1, 1970, 00:00:00 UTC

  `var myDate = new Date(1992, 12, 19, 12, 21, 12, 12);`

- the `getTime()` would return // `727464072012`

```
let myDate = new Date(1992, 12, 19, 12, 21, 12, 12);
let myDate2 = new Date(1992, 12, 19, 12, 21, 12, 12);
myDate === myDate2 // false
myDate.getTime() === myDate2.getTime() // true
```

javascript Timers

- `setTimeout()` is used to execute a function after a specified number of milliseconds
- `setInterval()` is used to execute a function repeatedly, with a fixed time delay between each call
- other timer function in Js are `setImmediate()` `clearTimeou t()` `clearInterval()` `clearImmediate()`
- set the timer to a variable to stop the timer later
  `var myTimer = setTimeout(function() { console.log('hello'); }, 3000);`

```
var myTimer = setTimeout(function() {
  console.log('hello');
}, 3000);
// after 3 seconds the console will print hello

var myTimer2 = setInterval(function() {
  console.log('hello');
}, 3000); // after interval of 3 seconds the console will print hello

var myTimer3 = setImmediate(function() {
  console.log('hello');
}); // immediate executes the function immediately

clearTimeout() // clear the timer
```

Local Storage in Js

- local storage is used to store data in the browser
- data can be stored in local storage using `localStorage.setItem()`
- data can be retrieved from local storage using `localStorage.getItem()`
- other local storage methods are `localStorage.removeItem()` `localStorage.clear()`

```
localStorage.setItem('name', 'John');
localStorage.setItem('age', '30');
console.log(localStorage.getItem('name'));
console.log(localStorage.getItem('age'));
```