7. Advanced JavaScript Features

7.1 For-Of Loops and Objects

- For-Of does not work on object
  - Object is not iterable
- Arrays are iterable

```
const alphabets = ["a", "b", "c"];
for (const alphabet of alphabets) {
  console.log(alphabet);
}

// a
// b
// c
```

- To iterate over objects, we need to use
  - Objects - Keys( )
  - Values ( )
  - Entries ( )

```
const person = {
  name: "Nikhil",
  age: 25,
  skill: "ReactJS",
};

for (const key of Object.keys(person)) {
  console.log(person[key]);
}

// Nikhil
// 25
// ReactJS
```

- For In vs For Off
  - For In - Not only iterates over the object but also the prototype
    - This makes it unreliable

```
const car = {
  engine: true,
  steering: true,
  speed: "slow",
};

const sportsCar = Object.create(car);
sportsCar.speed = "fast";
console.log("The SportsCar object: ", sportsCar);

for (prop in sportsCar) {
  console.log(prop);
}

// speed
// engine
// steering
```

  - Here, we can see that the properties of prototype are also counted

---

7.2 Template Literals

- Backticks can be used to define the value of strings.
- Main uses
  - Variable interpolation
  - Spanning string in different lines

- Expression evaluation

7.3 Data Structures

- Simple method to organize data
- **Object**
  - Unordered, Non-iterable : Collection of key-value pairs

```
// Converting Obj to Arr

const drone = {
  speed: 100,
  color: "yellow",
};
let result = [];
const droneKeys = Object.keys(drone);
droneKeys.forEach((key) => result.push(key, drone[key]));
console.log(result);

// Although, this can be done - but it means that we have not chosen the correct data
// structure
```

- **Array**
  - Ordered and Iterable collection of values that can be accessed using index
  - Methods on Array
    - for Each
      - We can use this function to run through all elements of Array by accessing the index-value

```
// Array

// For Each

const fruits = ["kiwi", "mango", "apple", "pear"];

const appendInd = (fruit, index) => {
  console.log(`${index} - ${fruit}`);
};

// fruits.forEach(appendInd);

fruits.forEach((fruit, index) => console.log(index, "-", fruit));
```

    - filter

```
const numbers = [10, 30, 50, 70, 5, 2, 3];
const result = numbers.filter((number) => number > 30);
console.log(result);

//[50, 70]
```

- map
  - returns a new array

```
const mapNumbers = numbers.map((number) => number * 2);
console.log(mapNumbers);
```

- **Map**
  - Iterable
  - Key-Value Pairs
  - Similar to object - but here, we can use any values as keys
  - In object → only Strings and Symbols
  - We can get the size of map but not objects
  - Doesn’t have any inheritance or prototype!

```
let rankings = new Map();
rankings.set(1, "Gold");
rankings.set(2, "Silver");
rankings.set(3, "Bronze");

console.log(rankings);
console.log(rankings.get(3));

// Map(3) { 1 => 'Gold', 2 => 'Silver', 3 => 'Bronze' }
// Bronze
```

- **Set**
  - Items need to be unique
  - Collection of unique values

```
const customers = [
  "Adam",
  "Bruno",
  "Clark",
```

```
  "Dwayne",
  "Clark",
  "Edgar",
  "Bruno",
  "Adam",
];
const uniqueCustomers = new Set(customers);
console.log(uniqueCustomers);

// Set(5) { 'Adam', 'Bruno', 'Clark', 'Dwayne', 'Edgar' }
```

7.4 Spread Operator

- **Spread** → used to un-pack an array
  - Concatenate two Arr

```
const initial = ["a", "b", "c"];
const latter = ["d", "e", "f"];
const combination = [...initial, ...latter];
console.log(combination);

// [ 'a', 'b', 'c', 'd', 'e', 'f' ]
```

  - Concatenate two Objects

```
const flying = {
  wings: 2,
};

const car = {
  door: 4,
};

const flyingCar = { ...flying, ...car };
console.log(flyingCar);

// { wings: 2, door: 4 }
```

  - Add new members to array

```
const originalArr = [1, 2];
const modifiedArr = [...originalArr, 3, 4];
const modifiedArr2 = [3, ...originalArr, 4];
console.log(modifiedArr);
console.log(modifiedArr2);
```

```
// [ 1, 2, 3, 4 ]
// [ 3, 1, 2, 4 ]
```

- Convert string to array → using Spread operator

```
const greeting = "Hello";
const arrayOfChars = [...greeting];
console.log(arrayOfChars);
```

- **Rest** → used to pack an array

```
const addTaxToPrices = (taxRate, ...itemsBought) => {
  return itemsBought.map((item) => taxRate * item);
};

let shoppingCart = addTaxToPrices(1.1, 2, 4, 6, 8);
console.log(shoppingCart);
```

- The Rest parameter → should always be the last parameter in definition

---

7.5 Modules

- JS Modules are standalone unites of code → can be re-used repetitively.
- In older versions of JS :
  - All functions that are defined on Windows object → tend to be global.
  - This was useful for simple project → but could cause issues when building complex projects.
  - Global function from one script could override function from another script.
  - Techniques were developed to bypass → but these were not without flaws.
- An engineer at Mozilla - Kevin Bangor - fixed this using a project : Server JS
  - Also known as, Common JS.
  - Common JS → Defines how modules should work outside of browser environment.
  - However, not all the keywords tend to work in browser environment.

7.6 JavaScript DOM (Document Object Model) Manipulation

- DOM Tree Structure
  - HTML file is mapped out like tree structure
  - Saved as JavaScript object in browser’s memory
- Remember some basic keywords
  - create Element - to create any element
  - inner Text - for initial text
  - set Attribute - for class and ID
  - document body append Child

7.7 JavaScript Selector

- document
- query Selector → Returns the first element
- query Selector All - Returns all element
- element by ID
- elements by Class Name
  - Remember elements in plural

```
let answer = prompt('What is your name?');
if (typeof(answer) === 'string') {
    var h1 = document.createElement('h1')
    h1.innerText = answer;
    document.body.appendChild(h1);
}
```

7.8 Event Handling

```
const target = document.querySelector('body');
const handleClick = () => {
  console.log('Clicked!')
}
target.addEventListener('click',handleClick);
```

- Event Listener on Example.Com

```
var h1 = document.querySelector('h1')
var arr = [
    'Example Domain',
    'First Click',
    'Second Click',
    'Third Click'
]
function handleClicks(){
    switch(h1.innerText){
        case arr[0]:
            h1.innerText = arr[1]
            break;

        case arr[1]:
            h1.innerText = arr[2]
            break;

        case arr[2]:
            h1.innerText = arr[3]
            break;

        default:
            h1.innerText = arr[0]
    }
}

h1.addEventListener('click',handleClicks);
```

7.9 JSON

- To access JSON in JS
    - We need to import it - use assert {type : ‘JSON’}
    - Also add type = ‘ module ‘ → on script tag
    - Then we need to use *fetch* function
    - We need to supply the complete server URL to fetch

```
// sample.json

{
  "greeting": "hello"
}

// index.html

<body>
    <script src="json.js" type="module"></script>
```

```
</body>

// json.js

fetch("http://127.0.0.1:5500/sample.json")
  .then((response) => response.json())
  .then((json) => console.log(json));
```

- Convert JSON to Regular Object

```
const jsonStr = '{"greeting" : "hello"}'
JSON.parse(jsonStr)

// {greeting: 'hello'}
```

- Convert Data to JSON Object

```
const data = {
    fName : "Nik",
    lName : "Tan"
}

JSON.stringify(data)

//'{"fName":"Nik","lName":"Tan"}'
```

```
const data2 = {
    fName : "Nik",
    lName : "Tan",
    collect : function(){console.log('hey')}
}

JSON.stringify(data2)
// '{"fName":"Nik","lName":"Tan"}'

// Functions are not convereted to JSON
```