Assesment 3

1. Select elment by its ID using JavaScript

```
Document.getElementByID("id-from-html")
```

2. Select Elements of specific class name

```
Document.getElementsByClassName("class-name-from-html")
```

3. Select the first child element of a given element

```
//store parent element in variable
let parentElement = Document.getElementById("parent-element-id-from-html")

// access first child by using firstElementChild
parentElement.firstElementChild
```

4. How can you select all elements that match a specific CSS selector

```
//use querySelectorAll
Document.querySelectorAll("h1") // will select all h1 tags
```

5. Change the text content of an element

```
//Grab element store in variable
let element = Document.getElementById("use-id-from-html")

//change the text content by using innertext
element.innerText = 'This is new content'
```

7. Modify value of HTML attribute

```
//use setAttribute method
element.setAttribute('attribute' , 'new value')
```

8. Add a new element to the Dom

```
// createElement method
let element = document.createElement("div") //will create a dive element
```

9. Remove element from the Dom

```
//use remove method
element.remove()
```

10. Acess Parent Element

```
let childElement = document.getElementById("child-element-id")

//access parent element
childElement.parentElement
```

11. Access next sibling element

```
//use nextElementSibling
siblingElement.nextElementSibling
```

12. Iterate over all child element of a given element

```
//first grab parent container
let parentContainer = document.getElementById("parent-container-id")

//access children property to get HTML COLLECTION
let childElements = parentContainer.children

//use for loop to iterate
for(let i=0; i<childElements.length; i++){
        console.log(childElements[i])
}
```

13. Find all elements matching a specific condition within subtree

```
//Find root element
let rootElement = document.getElementById("root-element-id")
//use querySelectorAll to find element matching condition
let matchingElements = rootElement.querySelectorAll('.specific-condition')
```

14. Change inline style of element in javascript

```
//use style property to acheive this
element.style.backgroundColor = "black"
element.style.color = "white"
```

15. Add & Remove Css classes from element

```
//use classList to acheive

//ADD Class
element.classList.add('name-of-css-class')

//REMOVE Class
element.classList.remove('name-of-css-class')
```

16. Toggle between 2 different styles for an element

```
//use toggle method inside an if else block

if (element.classList.contains("style1")){
        element.classList.toggle('style2')
} else {
        element.classList.toggle('style1')
}
```

17. Dynamically create and add Css stylesheet to document

```

```

18. Explain syntax and purpose of forEach method

```
//Modern built in javascript method for iterating over an array.
//Allows you to execute function one time for each element inside the array.

//syntax use for each with a call back function
arr.forEach(function(num, index){
        console.log(`Value @ ${index} position is ${num}`)
})

//use arrow function with forEach
arr.forEach(value => {
        console.log(value)
})
```

19. How does forEach loop difer from traditional for loop in scope and behavior

Syntax

- for each syntax is more concise and expressive. Will handle the iteration for you
- traditional for loop requires manual control, you have to initialize, provide a conditional statement and increment or decrement
  Scope
- forEach call back function has its own scope. variables declared within are scoped to function
- for loop variables declared within loop initialization are scoped to entire block.
  Behavior
- forEach loop cant use `break` keyword to terminate early, can use `return` to terminate the callback function but it wont break out of the forEach. Can't use `continue` to skip to next iteration but again can use `return` to accomplish this. Also iteration order is in ascending order.
- for loop can use `break` to terminate early. can also use `continue` to skip to next iteration. Iteration order can be controlled by you, you can do reverse iteration

20. Write code to double each element in an array using forEach.

```
arr[10, 20, 30, 40] // -> should result in [20, 40, 60, 40]

//multiply each value by 2. Using forEach method
arr.forEach((value, index) => {
        arr[index] = value * 2
        })
```

21. Create new array containing only strings from mixed array using forEach

```
//Declare mixed array
const mixedArray = [1, `Usman`, false, 'orange', 'Sofhia' ]
let newArray = [] //declare new empty array

//use for each to iterate through each value
mixedArray.forEach(value => {
        //compare if value in current iteration is a string or not
        if(typeof value === 'string'){
                //if value is a string push it to the new array
                newArray.push(value)
        }
})
```

22. Filter an array to include only even numbers using forEach

```
const originalArray = [2, 4, 3, 6, 10, 11, 15]
let newArray = []
originalArray.forEach(value => {
        if(value % 2 === 0){
                newArray.push(value)
        }
})
```

24. Demonstrate how to access the current index within a forEach callback

```
// Declare an example array to be used in forEach method
const arry = [12, 15, 18, 19, 3]

//use forEach method to iterate on array and access index and log out
arry.forEach((value, index) => {
        console.log(`At position ${index} the value is ${value}`)
})
```

25. Explain how to break out of a forEach loop early

`forEach` method does not have built in mechanism for this. It is built to iterate over the entire array. But workarounds can be use such as the `return` keyword inside the callback function.

26. Can you modify the original array directly within a forEach callback. Why or why not

Yes you can use the 'index' parameter from the callback function to modify the value for that specific index for e.g

`arr.forEach((item, index, array) => {` array[index]= item * 2
`})`

27. Describe common use cases for forEach in JavaScript applications

- Use case one where you need to iterate and print an entire array
- Use case 2 event handlng, to attach event listeners to multiple elements for eg:

```
let buttons = document.querySelectorAll('.button')

buttons.forEach(funtion(button){
        button.addEventListener(`click`, function(){
                console.log('Button clicked')
        })
})
```

28. When would you choose forEach over other looping method

- When working with arrays that are not too large. Its simple and readable use of call back function allows for more declarative code making your intentions clearer. When you want to avoid control flow complexity.

28. Explain the syntax and purpose of the for of loop in Javascript.

- Is a control flow statement allows you to iterate over iterable objects such as arrays, strings, maps, sets and more. Introduced in 2015
- makes it easy to loop through elements without needing to handle the index or iteration logic, makes code easier to read
  syntax

```
for(variable of iterable){
        //code block to be executed
}
```

29. How does for..of differ from traditional for loops and forEach

- for `of` loop are the more modern and concise option for iterating on an iterable object
- for `of` loop differs with traditional loop in that there is no need to manage indexing forOf loop handles that
- for `of` loop is more versatile can iterate over a broader range of iterable objects. `forEach` is specifically geared towards dealing with arrays

30. Iterate over an array using for of loop and print their squares

```
//declare an array with numbers
let numArray = [2, 4, 10, 12, 13]

//use forof loop to square the numbers
for(num of numArray){
        const square = Math.pow(num,2)
        console.log(square)
}
```

31. Create a new array containing the string lengths of each element in another array using for...of.

```
const orgArray = ["Usman","Qamar","USA","World"]
let orgArrayItemLength = []
```

```
for(str of orgArray){
        orgArrayItemLength.push(str.length)
}

console.log(`Length of each individual item in orgArray is listed in order :
${orgArrayItemLength}`)
```

32. Iterate over the keys of an object using for...of.

```
//create object
let obj = {
        name : "usman",
        age : 30,
        Country : "USA"
}

//iterate over keys using Object.keys() method
for(key of Object.keys(obj)){
        console.log(key)
}
```

33. Iterate over values of object using for of loop

```
const obj = {
        name : "Sofhia",
        age : "30",
        Country : "USA"
}

//Iterate over the values in obj using Object.values() method
for( const value of Object.values(obj)){
        console.log(value)
}
```

34. Explain how to access both keys and values in a for..of loop

```
const obj = {
        name : "Usman",
        age : 20,
        Country : "USA"
}

for(const [key, value] of Object.entries(obj)){
```

```
        console.log(`${key} : ${value}`)
}
```

35. Explain how to access both keys and values in a for..of loop?

- You use the `Object.entries()` method which gives you access to both the key and value inside the object. You can then use a for loop to iterate over this by storing it in a keyValue array by using the following syntax: `for(const [key, value] of Object.entries(obj)`

36. Can you modigy the original array or object within a for..of loop?
- Yes you can modify an Array but since you dont have direct access to index you can use the `Array.entries()` method to get both the value and the index example:

```
let arr = [1,3,6,8]
for(const [index, val] of arr.entries()){
        arr[index] = val + 2
}
console.log(arr) //Output : [3, 5, 8, 10]
```

- In an Object you cannot directly modify the properties within a for...of looop because the loop gives you the values, not keys. However if you have an array of objects then you can use for...of loop to modify because it gives you reference to each object in the array eg:

```
let arrayOfObjects = [{ id: 1, name: 'Alice' }, { id: 2, name: 'Bob' }]; for
(let obj of arrayOfObjects) { if (obj.id === 1) { obj.name = 'Updated Name';
} } console.log(arrayOfObjects); // Output: [{ id: 1, name: 'Updated Name'
}, { id: 2, name: 'Bob' }]
```

37. Describe common use cases for for...of loops in JavaScript applications.
The `for...of` loop in JavaScript is a powerful tool for iterating over iterable objects such as arrays, strings, maps, and sets. Provides a simple readable way to access each element in an array without haveing to deal with indices. Also working with DOM collections you often deal with collection of nodes which are iterable `for...of` loop can be used to iterate over these collections. Cannot be used to iterate over plain objects because they are not iterable by default. However you can use `Object.keys()`, `Object.values()`, or `Object.entries()` to create an array from an object, which can then be iterated over using the `for...of` loop

38 When would you choose for...of over other looping methods?
In summary, `for...of` loops are a good choice when you need a simple, readable, and concise way to iterate over elements of iterable objects, and when you want to avoid the

boilerplate code associated with managing indices and bounds checking. However, if you need to perform operations that require access to the index of the array or if you're dealing with non-iterable objects, you might need to use other looping constructs or methods such as `for` loops, `forEach`, or `Object.keys()`, `Object.values()`, and `Object.entries()` for objects