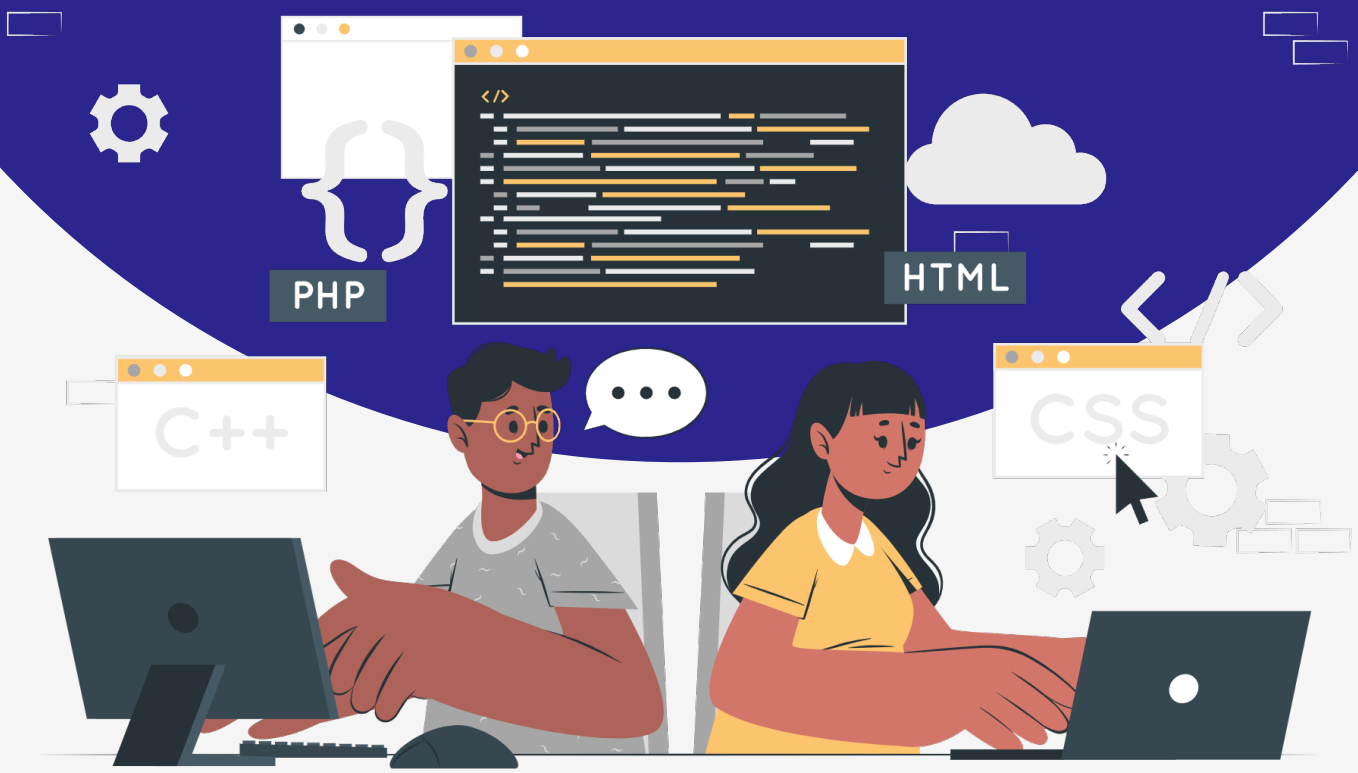
Lesson:

Arrays in JavaScript



PW SKILLS

Topics Covered

1. **What is an Array?**
2. **Why do we need arrays in JavaScript?**
3. **Declaration of Array.**
4. **Array index and storing.**
5. **Accessing elements in an array**
6. **Changing values in an array**
7. **Delete keyword in arrays & its problem**
8. **Iterating of an Array item**
9. **Interview Point**

What is an Array?

An array in JavaScript is a data structure that stores an ordered list of elements. It can hold elements of any data type, including numbers, strings, objects, and even other arrays. Arrays are a type of object in JavaScript and have a number of built-in methods for adding, removing, and manipulating elements.

Array length property - it is used to get the length of the array, ie **array.length**

Note - it starts counting from one.

Example -

```javascript
const language = ["javascript", "python", "java", "golang"]
const len = language.length
console.log("Array length =", len)
// Array length = 4
```

Example of an array:

```javascript
let player= ["Virat Kohli", "Rohit Sharma", "Suryakumar
Yadav", "KL Rahul", "Ravindra Jadeja", "Rishabh Pant",
"Shivam"];

let numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0];

let array = ["Hello", 20, true]
```

Why do we need Arrays in JavaScript?

Arrays in JavaScript are needed for several reasons:

1. **Grouping related data:** Arrays allow you to group related data together and make it easier to manipulate. For example, you could use an array to store a list of names or a list of products.

2. **Storing large amounts of data:** Arrays can store large amounts of data in a single variable, making it easier to manage and manipulate the data. This can be especially useful when working with large datasets.

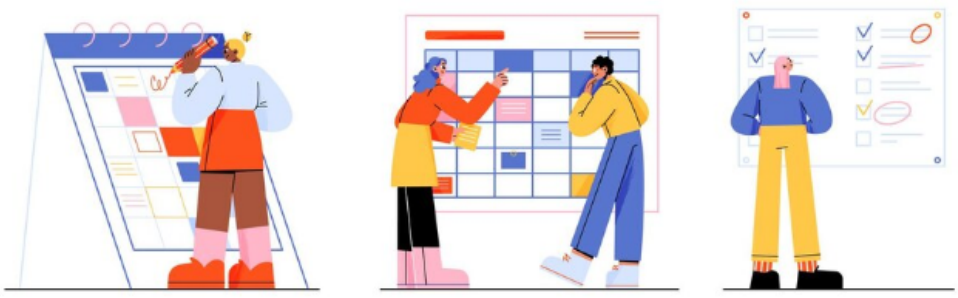
3. **Improving performance:** Arrays can improve performance when working with large amounts of data. Because the elements in an array are stored in consecutive memory locations, it is faster to access elements in an array than it is to access elements in other data structures, such as objects.

4. **Ease of use:** Arrays come with a number of built-in methods for adding, removing, and manipulating elements. This makes it easy to perform operations on the data stored in an array.

5. **Better readability:** Arrays make your code more readable by grouping related data together in a single structure. This makes it easier for others to understand your code and makes it easier for you to maintain your code in the future.

Overall, arrays in JavaScript are a crucial tool for managing and manipulating data in various locations. They provide a flexible and efficient way to store and process data and make writing clean, maintainable code easier.

Declaration of Array:



There are several ways to declare an array in JavaScript.

1. **Using square brackets [ ]:** This is the most common and recommended way to declare an array in JavaScript. You can create an array by enclosing a comma-separated list of elements in square brackets:

```javascript
let fruits = ["Apple", "Mango", "Banana", "Kiwi"];
```

2. **Using the Array constructor:** You can also create an array using the Array constructor. You can pass in the length of the array or a list of elements:

```
JavaScript
let numbers = new Array(1, 2, 3, 4, 5);

let emptyArray = new Array(5); // Creates an array with 5
empty elements.
```

It is generally not recommended to create an array using the Array constructor in JavaScript, due to the following reasons -

- Inconsistent behaviour - The behaviour of the Array constructor is inconsistent depending on the number and type of arguments passed to it. If you pass a single numeric argument, it is treated as the desired length of the array, and an empty array with that length is created. If you pass multiple arguments, those arguments become the elements of the array. This can lead to confusion and unexpected results.
- Simplicity and readability - Using an array literal notation ([]) to create an array is simpler and more readable. It is a widely adopted convention in JavaScript code and provides a clear and concise way to define arrays.
- Potential security risks - Using the Array constructor with a single numeric argument can lead to potential security risks if the value is not properly validated. If an attacker is able to supply a malicious value as the argument, it could result in unexpected behavior or vulnerabilities.

3. **Using an array literal:** This is a shorthand method for creating arrays that are equivalent to using square brackets:

```
JavaScript
let colors = Array("Black", "Red", "White", "Blue");
```

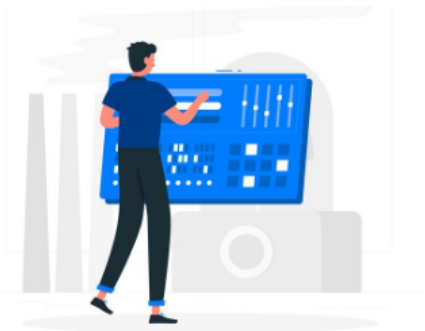
Array index and storing

Indexing and storing in arrays are related concepts in JavaScript. The index of an element in an array determines its position within the array while storing in an array refers to the act of assigning a value to an element within the array.

In JavaScript, arrays are zero-indexed, meaning that the first element has an index of 0, the second element has an index of 1, and so on. To store a value in an array, you can use the square bracket notation and the index number:

```javascript
let players= []; // create an empty array
fruits[0] = "Virat"; // store "Virat" at index 0
fruits[1] = "Rohit"; // store "Rohit" at index 1
fruits[2] = "Suryakumar"; // store "Suryakumar" at index 2
```

Accessing elements in an array



Accessing elements in an array in JavaScript is done using the square bracket notation and an index number. The index number represents the position of the element within the array, starting from 0 for the first element, 1 for the second element, and so on.

For example, consider the following array:

```javascript
let players = ["Virat", "Rohit", "Suryakumar"];
// To access the first element "Virat", you would use the
following code.

players[0]; // returns "Virat"
```

You can access any element in the array using its index number:

```javascript
players[1]; //returns "Rohit"
players[2]; // returns "Suryakumar"
```

It's important to keep in mind that if you try to access an index that is outside the bounds of the array, you will get undefined.

And also if you try to access a negative index it will give you an undefined.

For example,

```javascript
players[3]; // returns undefined
players[-1]; // returns undefined
players[-3]; // returns undefined
```

In some other programming languages like Java, C, and C++ if you try to access a negative index, you will get an error which is ArrayIndexOutOfBoundException or Invalid OutOfRange.

Overall, accessing elements in an array is a basic operation in JavaScript that allows you to retrieve values stored in the array. Understanding how to access elements in an array is essential for working with arrays in JavaScript.

Changing values in an Array

In JavaScript, An Array value item can be changed by accessing the element with its index and assigning a new value to it.
Example:

```javascript
const hobbies = ["Coding", "Learning", "Watching movies"]

hobbies[0] = "building Project" // change to building Project
hobbies[2] = "Football"  // change to Football

// output -
[ 'building Project', 'Learning', 'Football' ]
```

The above example is used to change the value of an array using the index of the particular array items.

Delete keyword in arrays & its problem

In JavaScript, the delete keyword is used to delete a property from an object, including an array element. However, using delete to remove elements from an array can lead to unexpected behaviour and introduce problems.

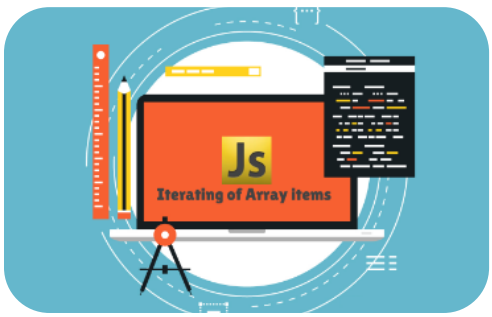
Here are some impacts and problems associated with using the delete keyword on arrays:

- **Sparse Arrays** - The delete keyword does not update the length property of the array. Therefore, if you delete an element using delete, the length of the array remains the same, giving a false impression of the array's actual size.
- **Array length** - The delete keyword does not update the length property of the array. Therefore, if you delete an element using delete, the length of the array remains the same, giving a false impression of the array's actual size.
- **Performance Impact** - Deleting an element with delete does not reclaim the memory associated with that element. It simply marks the element as undefined
- **Array Integrity** - Using delete on an array can disrupt the integrity of the array's order and indices. The deleted element is replaced with an empty slot, and subsequent elements are not automatically shifted to fill the gap

Example - delete in an array -

```JavaScript
const array = [1, 2, 3, 4];
console.log(array.length); // output - 4
delete array[2];
console.log(array); // output - [ 1, 2, <1 empty item>, 4
]
console.log(array[2]); // output - undefined
console.log(array.length); // output - 4 same length even
after delete
```

Iterating of Array items



In JavaScript, An array items can be iterated or looped in the following ways:

- **for loop**
- **for....of loop**
- **while loop**
- **forEach()**

for loop

It allows us to iterate over an array by using the index of the array items.

Example -

```javascript
const array = [1, 2, 3];

for (let index = 0; index < array.length; index++) {
  const element = array[index];
  console.log(element);
}
// - output -
1
2
3
```

for... of loop

It is a newer JavaScript feature introduced in ECMAScript 6 (ES6) that allows us to iterate over iterable arrays, including objects.

Example -

```javascript
const array = [1, 2, 3];
for (const iterator of array) {
  console.log(iterator);
}
// output ---
1
2
3
```

while loop

The while loop allows us to loop over an array by checking a condition before each iteration.

Example -

```javascript
const array = [1, 2, 4, 3];

let i = 0;
while (array.length > i) {
  console.log(array[i]);
  i++;
}
// output -
1
2
3
4
```

Interview Point

Q. Explain the purpose and usage of the delete keyword in JavaScript arrays. Can you highlight a potential issue with using delete?

Ans. The delete keyword in JavaScript arrays is used to remove an element at a specific index. However, it doesn't actually remove the element; instead, it leaves an undefined gap at the specified index. A potential issue is that it doesn't update the array length, and iterating over the array may encounter unexpected undefined values.

Q. Explain the role of the "const" keyword when working with arrays in JavaScript. How does using const impact common array operations like adding or removing elements? Explain with an example.

Ans. In JavaScript, the const keyword is used to declare constants, indicating that the assigned value cannot be reassigned. However, when const is applied to arrays, it doesn't make the array itself immutable. The reference to the array is constant, meaning we cannot reassign the variable to a new array, but the array elements can be modified.

Example:

```
const arr1 = [1,2,3,4];
console.log(arr1, 'length of this array is' , arr1.length);

arr1[4] = 12;
console.log(arr1, 'length of this array is' , arr1.length);

// output
[ 1, 2, 3, 4 ] length of this array is 4
[ 1, 2, 3, 4, 12 ] length of this array is 5
```

As we can see in the above example we can modify the array.

But when we try to use the same variable name for another array, then javascript will throw us an error

Example:

```
const arr1 = [1,2,3,4]

arr1[4] = 3

let arr1 = [1,2,3]
const arr1 = [1,2,3,4]

arr1[4] = 3

let arr1 = [1,2,3]
```

output:

```
Uncaught SyntaxError: Identifier 'arr1' has already been declared
```

Interview Point

Q. Explain the purpose and usage of the delete keyword in JavaScript arrays. Can you highlight a potential issue with using delete?

Ans. The delete keyword in JavaScript arrays is used to remove an element at a specific index. However, it doesn't actually remove the element; instead, it leaves an undefined gap at the specified index. A potential issue is that it doesn't update the array length, and iterating over the array may encounter unexpected undefined values.

Q. Write a JavaScript function that takes an array of numbers as an argument and returns the largest element in the array

Ans.

```
function findMax(arr) {
  let max = arr[0]; // Assume the first element is the largest

  for (let i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }

  return max;
}

const numbers = [12, 5, 8, 24, 3];
console.log(findMax(numbers));

// Output: 24
```

PW SKILLS

THANK YOU !