Class 04 – ES6

ADVANCE JAVASCRIPT

MERN STACK DEVELOPMENT - MODULE B

Today’s Topics

- Array methods (loops)
  - filter
- Lexical Scoping and Closures
- Synchronous JavaScript
- Asynchronous JavaScript

filter – Array Method

The **filter** method is a fundamental array method in JavaScript designed to create a new array containing elements that pass a specific test. It offers a concise and expressive way to extract elements from an existing array based on a provided callback function, which evaluates each element against a specified condition.

The syntax of the filter method is simple and intuitive. It takes a callback function as its argument, which should return a Boolean value. Elements that satisfy the given condition (where the callback function returns true) are included in the new array, while those that do not pass the test are excluded.

syntax

```
const array = [1, 2, 3, 4, 5]

const newArray = array.filter(element => {
return number % 2 === 0;
});

//[2, 4, 6, 8, 10]
```

Lexical Scoping and Closures

Lexical Scoping

Lexical scoping refers to the way the scope of variables is determined by their location within the code. In other words, a variable in a nested function can access variables in its own scope, as well as variables in the scope of the function that contains it.

Closures

A closure is created when a function is defined inside another function, allowing the inner function to access variables from the outer function even after the outer function has finished executing.

Lexical Scoping and Closures examples

Lexical Scoping example

example

```
function outerFunction() {
  let outerVariable = 'I am from outer';
  function innerFunction() {
      let innerVariable = 'I am from inner';
      console.log(outerVariable);
      // Accessing outerVariable from the outer scope
  }
  innerFunction();
}
outerFunction();
```

Closures example

example

```
function outerFunction() {
    let outerVariable = 'I am from outer';
    function innerFunction() {
        console.log(outerVariable);
        // Accessing outerVariable from the outer scope
    }
     return innerFunction;
}
const closureExample = outerFunction();
closureExample();
// The inner function still has access to outerVariable
```

Synchronous JavaScript

Synchronous JavaScript refers to the default behavior of the language where code is executed sequentially, one statement at a time, blocking further execution until the current operation is completed. In synchronous code, each statement is executed in order, and the program waits for each operation to finish before moving on to the next one.

example

```
const synchronous_function = () => {
   console.log('Start');
   // Blocking operation (synchronous)
   for (let i = 0; i < 3; i++) {
       console.log(i);
   }
   console.log('End');
 };
synchronousFunction();
```

Asynchronous JavaScript

Asynchronous JavaScript is a programming paradigm that allows code to execute independently of the main program flow. In asynchronous programming, operations don't block the execution of subsequent code, enabling the program to perform multiple tasks concurrently without waiting for each one to complete.

example

```
console.log('Start');

// Asynchronous operation using setTimeout
 setTimeout(() => {
console.log('Asynchronous operation completed after 2 seconds');


}, 2000);


console.log('End');
```

That's it for today!

Keep coding, stay curious, and enjoy your JavaScript journey!