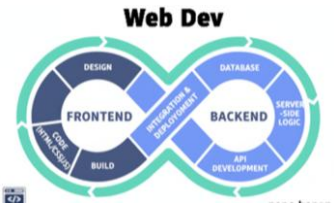
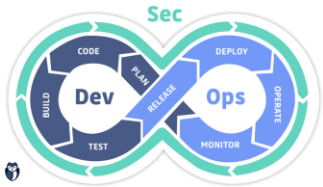




Day 41

"Web Development + Security"

Advanced JavaScript :

What is an IIFE?

IIFE stands for Immediately Invoked Function Expression. It's a function in JavaScript that runs automatically as soon as it's defined — without being called separately.

Syntax:

```
(function () {

  console.log("I run automatically!");

})();
```

Why use IIFE?

IIFEs are used to:

1. Avoid polluting the global scope (variables stay private)
2. Execute code immediately
3. Create a local scope for variables

A basic example of immediately invoked function:

Index.html:

index.html > html > body > script

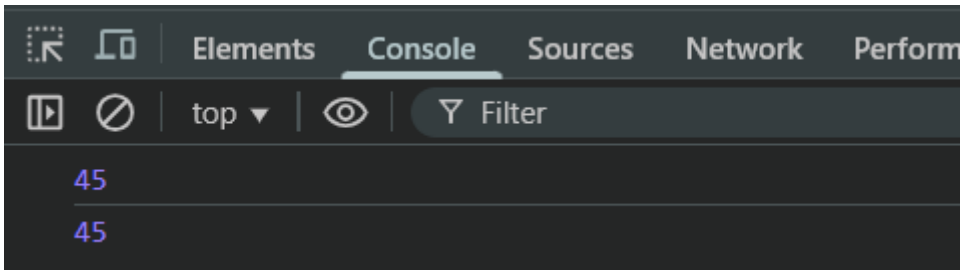
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="script.js"></script>
</body>
</html>
```

Script.js:

script.js > main

```js
async function sleep(){
    return new Promise((resolve,reject)=>{
        setTimeout(() => {
            resolve(45);
        },1000);
    })
}
(async function main(){
    let a = await sleep();
    console.log(a);
    let b = await sleep();
    console.log(b);
})()
```

Console:



What is Destructuring?

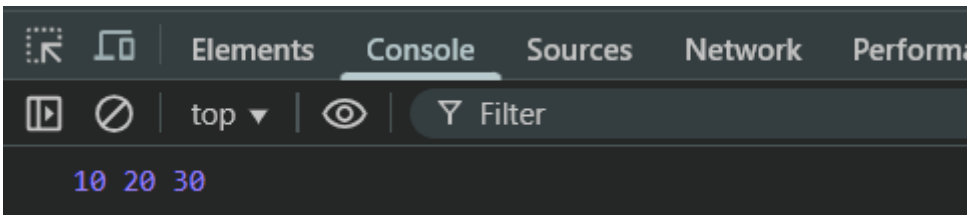
Destructuring means unpacking values from arrays or objects into separate variables —in a clean and short way. Instead of manually accessing each property or index, you can extract them directly.

Let's understand with the help of array:

Without destructuing:

```
15  //without destructuring
16  let numbers = [10, 20, 30];
17  let a = numbers[0];
18  let b = numbers[1];
19  let c = numbers[2];
20
21  console.log(a, b, c); // 10 20 30
```

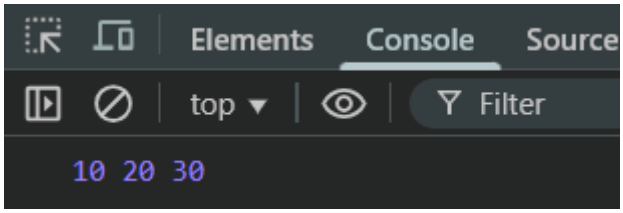
Console:



With Destructuring:

```
23  //With destructuring
24  let numbers = [10, 20, 30];
25  let [a, b, c] = numbers;
26
27  console.log(a, b, c); // 10 20 30
```

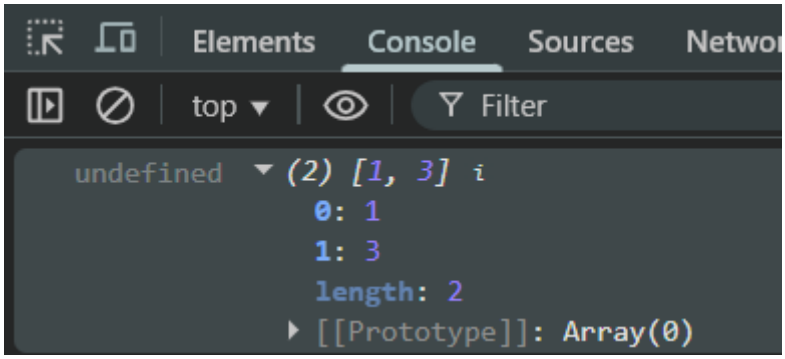
Console:



Similarly, IIFE without structuring:

```
29  //IIFE without structuring
30  async function sleep(){
31      return new Promise((resolve,reject)=>{
32          setTimeout(() => {
33              resolve(45);
34          },1000);
35      })
36  }
37  (async function main(){
38      let x,y = [1,3];
39      console.log(x,y);
40  })()
```

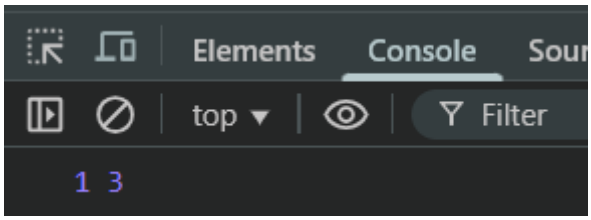
Console:



Now, IIFE with structuring:

```
42  //IIFE with structuring
43  async function sleep(){
44      return new Promise((resolve,reject)=>{
45          setTimeout(() => {
46              resolve(45);
47          },1000);
48      })
49  }
50  (async function main(){
51      let [x,y] = [1,3];
52      console.log(x,y);
53  })()
```

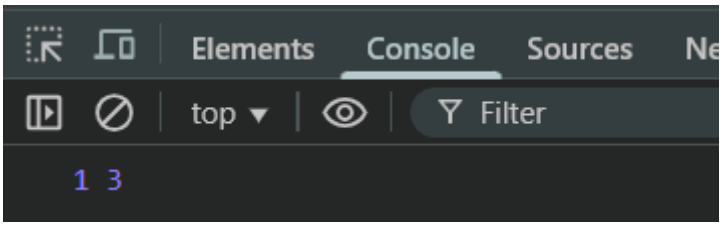
Console:



Now, In case we want give third variable value and have not decided in which it will be stored, then it will behave like this:

```
42  //IIFE with structuring
43  async function sleep(){
44      return new Promise((resolve,reject)=>{
45          setTimeout(() => {
46              resolve(45);
47          },1000);
48      })
49  }
50  (async function main(){
51      let [x,y] = [1,3,5];
52      console.log(x,y);
53  })()
```

Console:

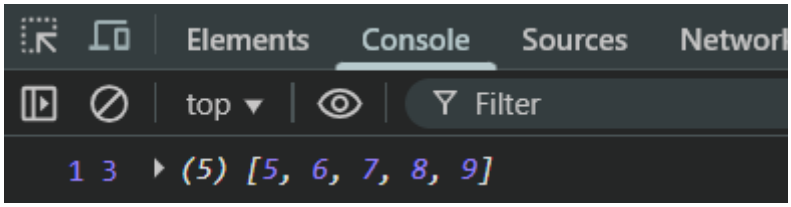


In order to tackle this issue, we need to use “rest” in this as show below:

Script.js: here the 5,6,7,8,9 are stored in ‘rest’

```
42   //IIFE with structuring
43   async function sleep(){
44       return new Promise((resolve,reject)=>{
45           setTimeout(() => {
46               resolve(45);
47           },1000);
48       })
49   }
50   (async function main(){
51       let [x,y, ... rest] = [1,3,5, 6, 7, 8, 9];
52       console.log(x,y,rest);
53   })()
```

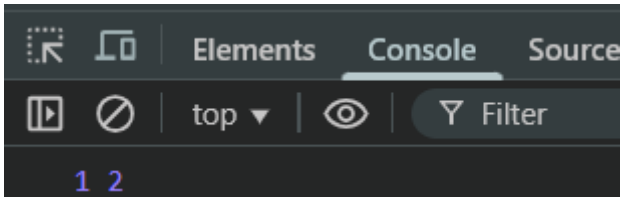
Console:



Also, we can destructure in the object as well:

```
55   async function sleep(){
56       return new Promise((resolve,reject)=>{
57           setTimeout(() => {
58               resolve(45);
59           },1000);
60       })
61   }
62   (async function main(){
63       let obj = {
64           a: 1,
65           b: 2,
66           c: 3
67       }
68       let {a,b} = obj;
69       console.log(a,b);
70   })();
```

Console:



What is Spread Syntax?

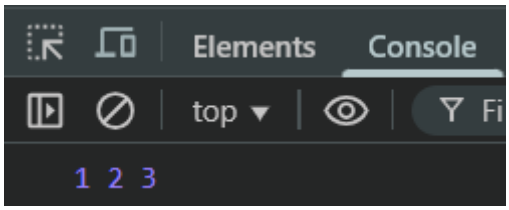
The spread syntax (...) is used to expand (or “spread out”) the contents of an array, object, or string into individual elements. It helps copy, combine, and pass values easily.

Example: a very basic example

Script.js

```
//Spread syntax
let arr = [1, 2, 3];
console.log(...arr); // 1 2 3
```

Console:



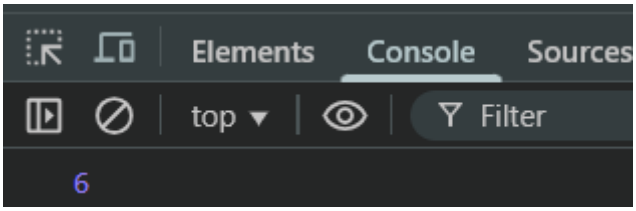
Example: adding the elements of array using the spread syntax

Script.js:

```
function add(a, b, c) {
  return a + b + c;
}

let numbers = [1, 2, 3];
console.log(add(...numbers)); // 6
```

Console:



What is Hoisting?

Hoisting means JavaScript moves declarations (not initializations) to the top of their scope — *before the code executes*.

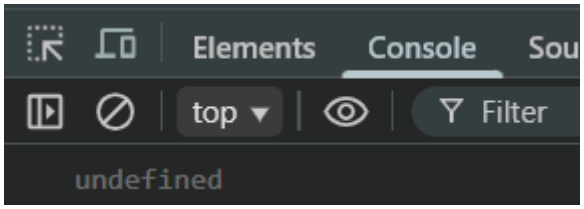
In simple words: You can use variables or functions before declaring them, because the declaration is "hoisted" (lifted) by JavaScript internally.

Example: var hoisting

Script.js:

```
83  //Hoisting
84  console.log(a); // undefined (not error)
85  var a = 10;
```

Console:

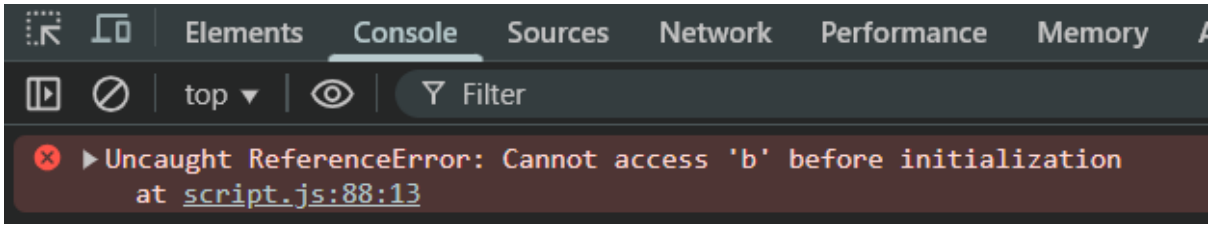


Example: let and const are not fully hoisted

Script.js:

```
86
87  //let hoisting
88  console.log(b); // ❌ ReferenceError
89  let b = 20;
```

Console:

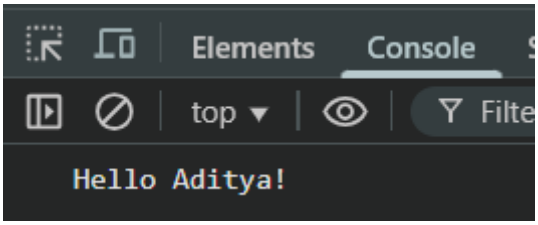


Example: Function declarations are fully hoisted — you can call them before defining them.

Script.js:

```
91  //function hoisting
92  greet(); // Works fine
93
94  function greet() {
95    console.log("Hello Aditya!");
96  }
```

Console:



--The End--