Lesson:

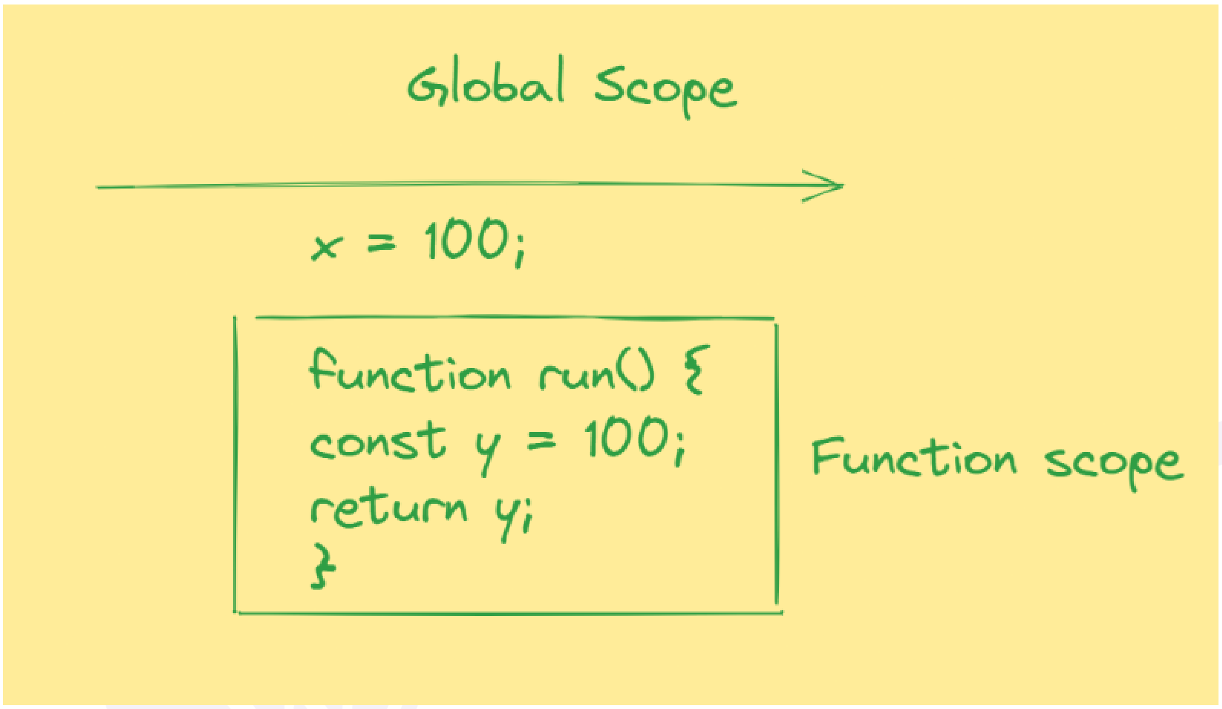
Scope

Topics Covered

1. Introduction to scope in js
2. Types of scope: global & local(functional & block)

Introduction to scope in js

Scope is an important concept, not only in JavaScript, but in many other programming languages. Scope is the current area or context of a specific piece of code. There are certain rules for what is accessible in specific scopes.



In JavaScript, when we write code in the **global scope** it is available everywhere, including functions. If we are NOT inside of a function or any kind of block, such as an if statement or a loop, then we are in the **global scope.**

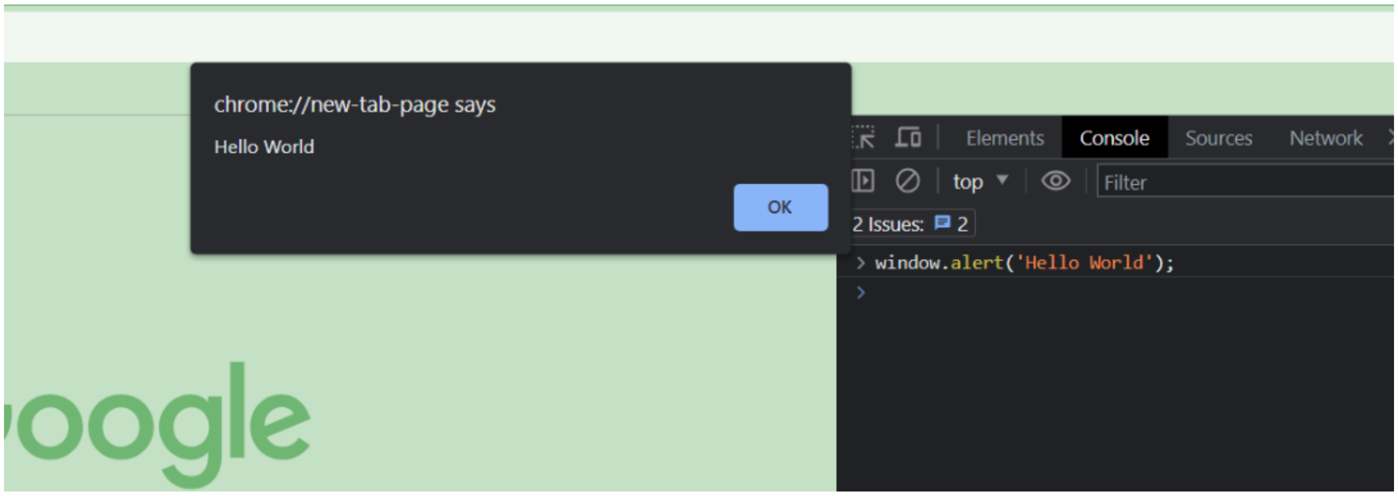
The `window` object

The browser creates a global object called **window.** This object has a ton of methods and properties on it that are available to us that we'll be looking at later.

The alert method belongs to the window object.

```JavaScript
window.alert('Hello World');
```

Output:



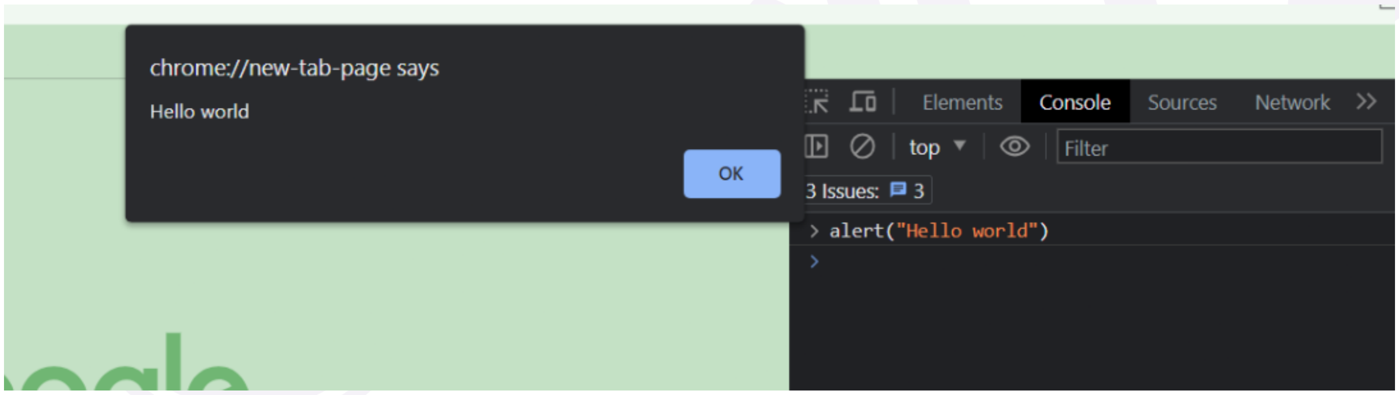
chrome://new-tab-page says

Hello World

```
> window.alert('Hello World');
>
```

Since the **window** object is the most top-level object in the browser environment, we don't need to use **window. We get the same output.**

JavaScript
```javascript
alert('Hello world');
```



chrome://new-tab-page says

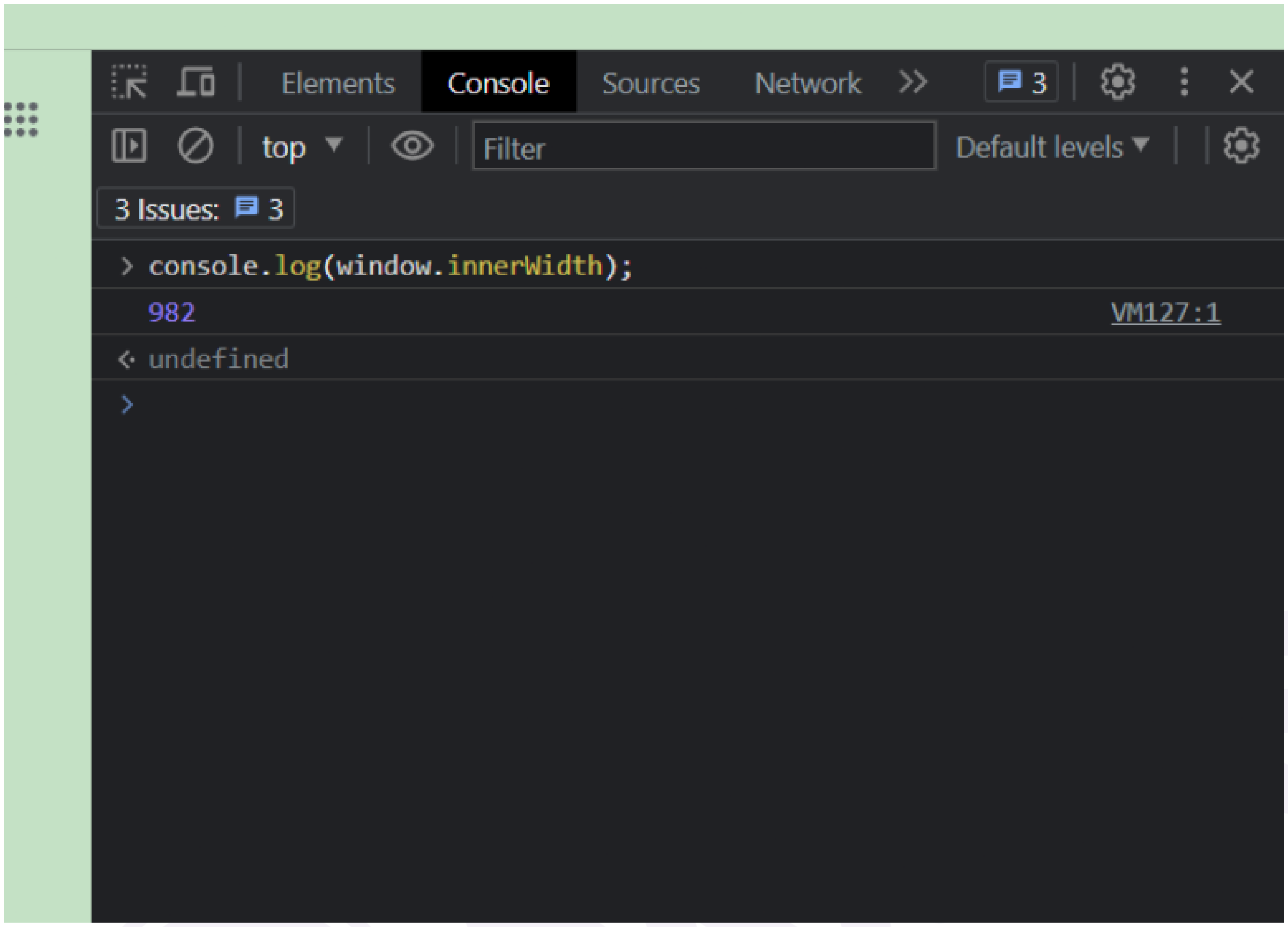
Hello world

```
> alert("Hello world")
>
```

innerWidth: It provides the width of the viewport through which the web page content is visible.

There is an **innerWidth** property on the window object. We could use that anywhere as well.

JavaScript
```javascript
console.log(window.innerWidth)
```



```
> console.log(window.innerWidth);
  982                                                    VM127:1
<· undefined
>
```

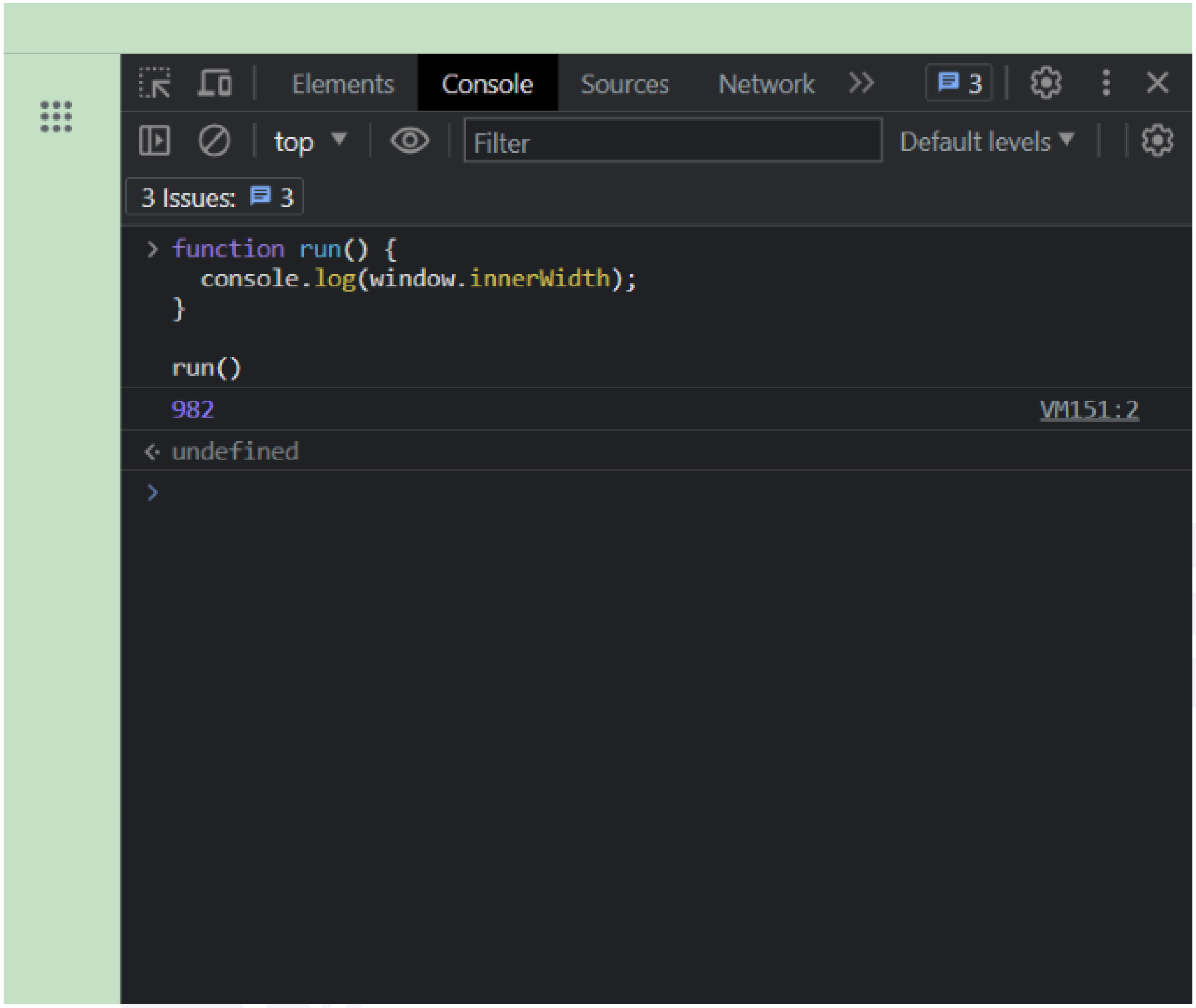
Note: innerwidth value will change according to the screen size..

We can use it in a **function** as well because it is **global.**

```javascript
function run() {
 console.log(window.innerWidth);
}

run()
```

Output:



```
> function run() {
    console.log(window.innerWidth);
  }

  run()
  982                                                    VM151:2
<· undefined
>
```

Note: The value of window.innerwidth will vary according to the size of the screen width.

Creating globally scoped variables
If we simply create a variable at the top of a JavaScript file, this is a **global** variable in the **global scope** and we can access it **anywhere**.

```JavaScript
const x = 100;
console.log(x); // 100
```

If we try to access x in the `run()` function, we can because it's **global.**

```
JavaScript
function run() {
 console.log(x); // 100
}
```

Local Scope
Local scope is created when functions and variables are only accessible with any function or block, hence we have two subtypes of Scopes

- Function Scope
- Block Scope

Function Scope
Function scope is the scope that is available to all code inside of a function. Any variables we define here will be available only inside of the function.

```
JavaScript
function add() {
  const y = 50;
}

console.log(y); // ReferenceError: y is not defined
```

Since **x** is **global**, We could use that in the add() function

```
JavaScript
function add() {
  const y = 50;
  console.log(x + y); // 150
}
```

Variable shadowing
If we create a variable called x in the function, it will overwrite the global variable and we can no longer access it. This is called variable shadowing.

```
JavaScript
const x =100;

function add() {
  const x = 1;
  const y = 50;
  console.log(x + y); // 51
}
```

Block Scope
Block scope is the scope that is available to all code inside of a block. A block is something like an **if** statement or any kind of **loop.**

Block scope was introduced in JavaScript with the introduction of the **let and const** keywords in ECMAScript 6 (ES6).

A block scope is created within any pair of curly braces {} (e.g., if statements, loops, functions). Variables declared with **let or const** are limited to the block scope and are not accessible outside of curly braces.

Example-1

```
JavaScript
const x = 100;

if (true) {
 console.log(x); // 100
 const y = 200;
  console.log(x + y); // 300
}

console.log(y); // ReferenceError: y is not defined
```

As you can see, we can not access **y** in the **global** scope because it belongs to the if statement block.

Example-2

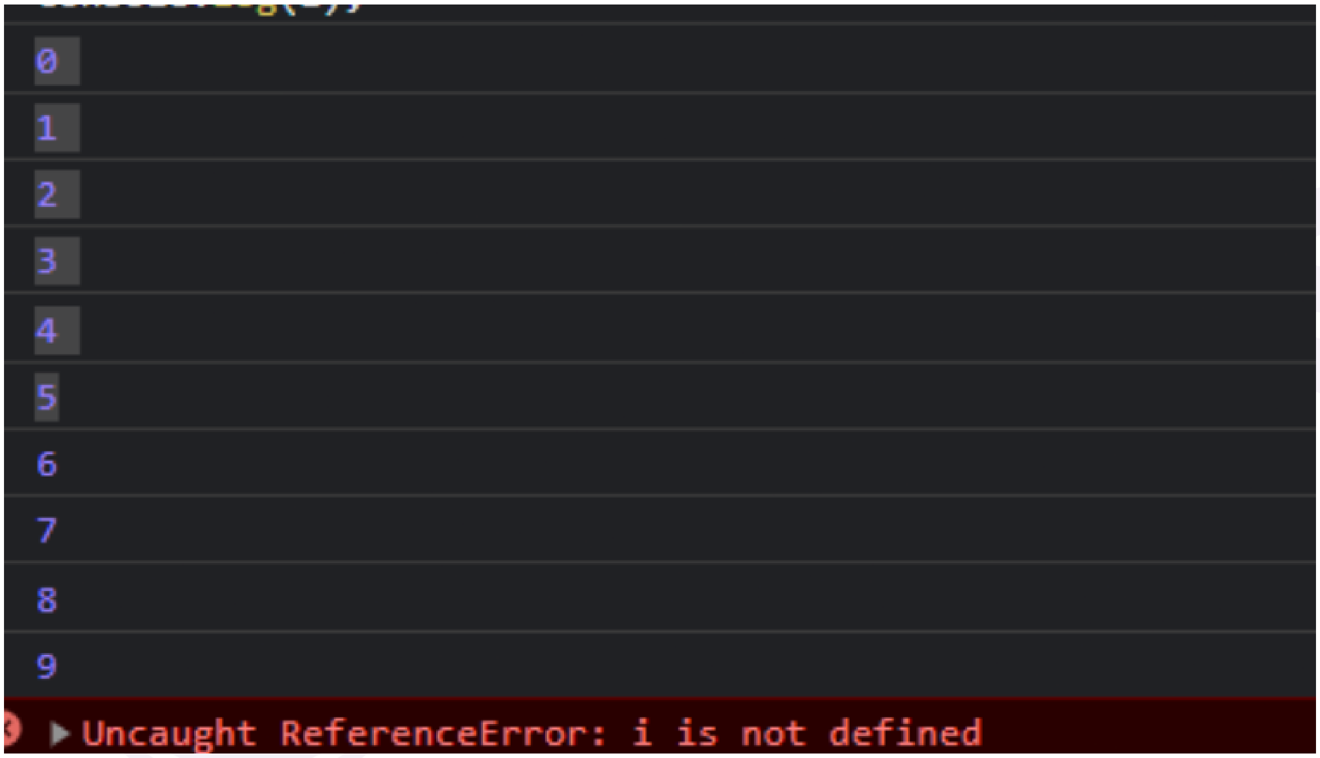
```
JavaScript
if (true) {
 var x = 10; // var has function scope
 let y = 20; // let has block scope
 const z = 30; // const has block scope
 console.log(x, y, z); // Output: 10 20 30
}
console.log(x); // Output: 10
console.log(y); // Error: y is not defined
console.log(z); // Error: z is not defined
```

Loop Example

I want to show you that loops do have their own block scope.

```JavaScript
for (let i = 0; i < 10; i++) {
 console.log(i);
}
console.log(i);  // ReferenceError: i is not defined
```

Output:



```
0
1
2
3
4
5
6
7
8
9
▶ Uncaught ReferenceError: i is not defined
```

As you can see, **'i'** is only available inside of the loop.