Introduction to JavaScript

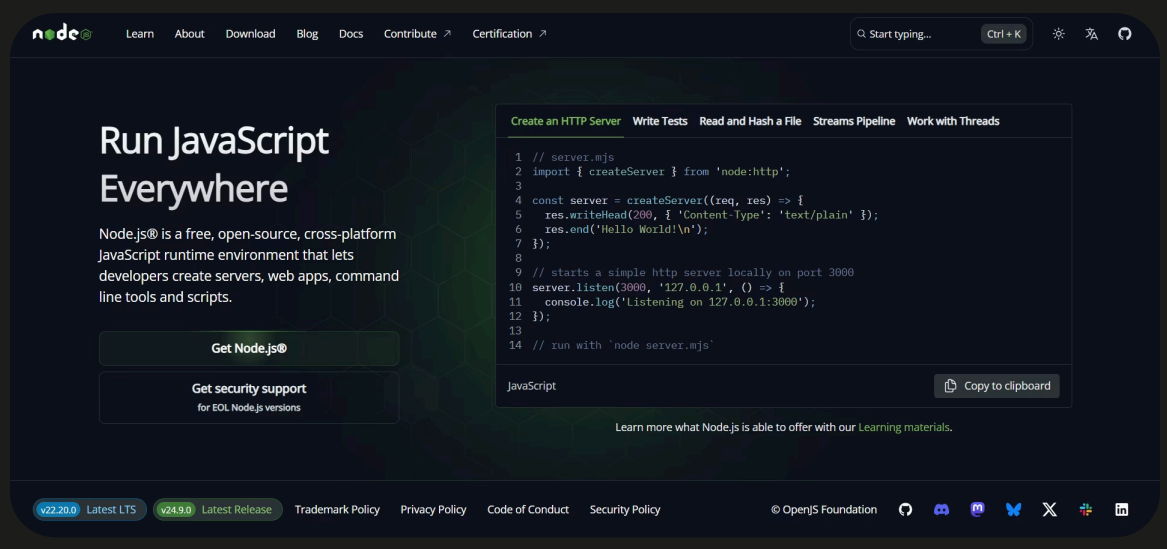
Lec-1

What is JavaScript ?

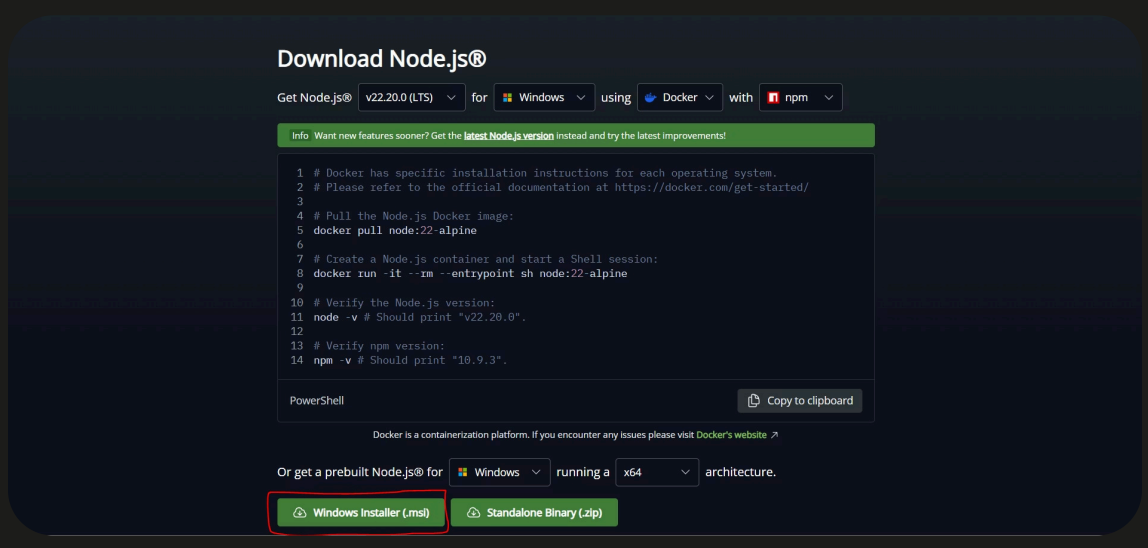
- A **programming language** mainly used for making web pages **interactive and dynamic**.
- Runs in the **browser (client-side)**, but can also run on **servers** with **Node.js (Runtime Environment of JavaScript)**
- Despite the name, **JavaScript is not Java**, They are completely different languages with different use cases.
- Code runs inside a **JavaScript engine** (e.g., V8 in Chrome, SpiderMonkey in Firefox).
- It is **interpreted**, not compiled like C++ or Java.
- Executes **line by line (but can optimize in background)**.
- It is a **Weakly Typed** Language **(No need to declare datatype).**

Setting up for JS

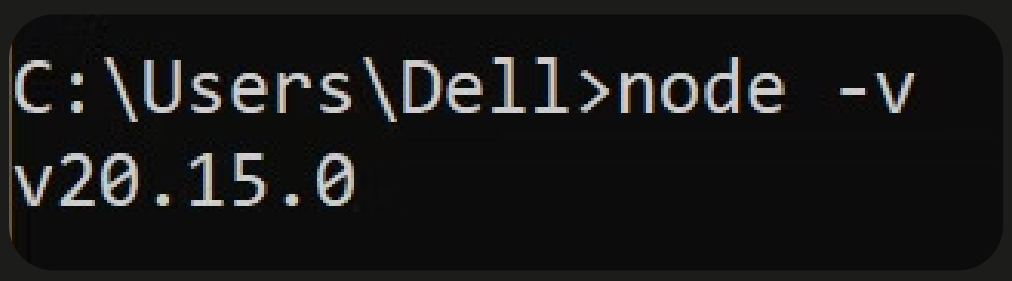
- Installing **NodeJS** is important to run JS, outside our browser.



- Visit the following URL: **https://nodejs.org/en/download**, and click on **Windows Installer (.msl).**



- After Successful installation, open **command prompt** and **run the following command** and match the expected output as shown in the following image.



- If it is showing you **any version of node.js**, that means **Your Installation was Successful.**

Helpful VS Code extension

Linking JavaScript to HTML

- There are **two** primary methods to link JavaScript to HTML.

1. Embedding JavaScript Directly in HTML:

- This method is **suitable for small, simple scripts that are specific to a particular HTML page.**
- Use the `<script>` tags: Place your JavaScript code directly within `<script>` and `</script>` tags in your HTML document.
- **Eg:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Intro to JS</title>
</head>
<body>
    <h1>Intro to JS</h1>
    <script>
        // Your JavaScript code goes here
    </script>
</body>
</html>
```

- You can place embedded JavaScript in either the `<head>` or `<body>` section.
- Placing it in the `<head>` **means it will execute before the page content,** while placing it in the `<body>` (ideally just before the closing `</body>` tag) **allows it to interact with the fully loaded HTML.**

2. Linking an External JavaScript File (recd):

- This is the **recommended** and most common approach for larger projects as it promotes code organization and reusability.
- **Create a JavaScript file** Create a new file with a `.js` extension (e.g., `script.js`).
- **Write your JavaScript code** in this file.
- **Link the file in your HTML** in your HTML file, use the `<script>` tag with the `src` attribute to point to the path of your JavaScript file.
- It is generally recommended to **place the `<script>` tag just before the closing `</body>` tag.** This ensures that the **HTML content is loaded and rendered before the JavaScript attempts to interact with it**, preventing potential errors.
- **Eg:**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Intro to JS</title>
</head>
<body>
    <h1>Intro to JS</h1>
    <!--

    Your HTML Code goes here

    -->
    <!-- Javascript link using script tag and src attribute -->
    <script src="./script.js"></script>
</body>
</html>
```

Hello World in JavaScript

- In JavaScript, anything can be **printed** or **logged** in the following way:

- A console is a panel that displays important messages like errors. If we want to see things by them appearing on our screen, we can print/log to our console directly.
- In JavaScript, the **console** keyword **refers to an object, a collection of data and action/method**, that we can use in our code.
- Keywords are nothing but **Interpreter aware words.**
- One action or method that is built into the console object is the **.log()** method. When we write **console.log()** what we put **inside a parenthesis will get printed or logged to the console.**
- Eg:

```
console.log(5) // Will log 5 to the console
```

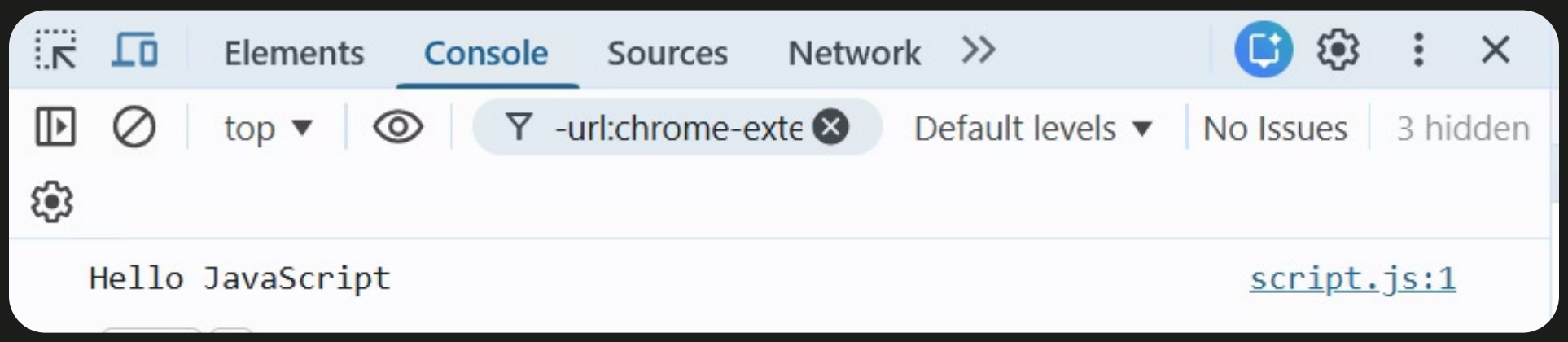
Ways to see output in JavaScript

- **Way-1 :** Open Terminal in VS Code (make sure you open it in same folder in which .js file is) and type the following command.

```
node script.js
```

> **Note:** after node, you should write your file name, here **script.js** is written considering our existing .js file name is **script.js**.

- **Way-2:** Link your .js file to an .html file creating both of them inside a same folder and open .html file using **live server (VS Code extension)**, after that open the **inspect tool** and **navigate to console tab** and see the output.



Variables in JavaScript

- In programming, a variable is a **container for a value**.
- Variables also provide a way of **labeling data with a descriptive name**, so our programs can be understood more clearly by the reader and by ourselves.

```
a = "John"
console.log(a) // This will print John
a = 25
console.log(a) // This will print 25
```

Above way of defining variables is not recommended in JavaScript

"use strict" directive

- The **"use strict"** directive was new in ECMAScript version 5.
- It defines that JavaScript code should be executed in "strict mode".
- It is not a statement. It is a literal expression, ignored by earlier versions of JavaScript.
- It should **always be declared at the beginning of your JavaScript file**.

```
"use strict" // declared at the top of the Js file
a = "John"
console.log(a)
```

Above lines of code will result in **Uncaught ReferenceError** : a is not defined

Creating a variable using "var" keyword

- **Var,** short for variable is a JS keyword that **creates** or **declares** a new variable.
- **Eg:**

```
var myName = "Mike"
console.log(myName) // This will print Mike
myName = "Micheal"
console.log(myName); // This will print Micheal
```

- If we declare a variable using **var** keyword and **left it unassigned with any value**, it will result as **"undefined"** in the console.
- **Eg:**

```
var unAss
console.log(unAss) // This will print undefined
```

Creating a variable using "let" keyword

- The **let** keyword signals that the **variable can be assigned a different value**.
- **Eg:**

```
let meal = "BigMac"
console.log(meal) // This will print BigMac
meal = "Burrito"
console.log(meal) // This will now print Burrito
```

- Similar to var, if we create a a variable using **let** keyword and **left it unassigned with any value**, it will result as **"undefined"** in the console.
- **Eg:**

```
let unAsslet
console.log(unAsslet); // This will print undefined
```

Creating a variable using "const" keyword

- Variables are declared using **const** in a same way as we declare **let** and **var.**
- **Eg:**

```
const constVar = "Julie"
console.log(constVar); // This will print Julie
```

- Using **const**, it is **not possible to declare a variable without assigning a value** to it.
- **Eg:**

```
const unAssconst // This will result in error - 'const' declaration must be initialised.
```

- Using **const**, it is also **not possible to reassign a new value to a variable which is already been declared and assigned** some value.
- **Eg:**

```
const constVar = "Julie"
constVar = "Mia"
console.log(constVar);
```

Above lines of code will result in **Uncaught TypeError**: Assignment to constant variable.

Note: What to use when ?

- While trying to decide between which keyword to choose while declaring your variable, remember:
- **use "let":** If you wish to reassign it's value later on
- **use "const":** If you do not wish to reassign a new value letter on
- Avoid using **"var"** due to issue of block scope or functional scope.

Datatypes

- Datatypes are **used to indicate the type of data** that we use in programming.
- Datatypes in JS are classified into 2 types based on- **how they're stored in a memory** and **whether they're of fixed size or not**, the types are as follows:
- **Primitve Datatypes:** These are of **fixed size** and they're **directly stored into stack memory**.
- **Non-Primitive Datatypes:** These are not of **fixed size** and their **data is stored in heap memory** and **reference is stored in stack memory**.

Primitive Datatypes

1. **number:** These includes any number, including number with decimals.

Eg:

```
let num1 = 25
console.log(num1); // This will print 25
num1 = 3.14
console.log(num1); // This will print 3.14
let num2 = 27.63
console.log(num2); // This will print 27.63
```

Note: Not-a-Number (NaN)

- In JavaScript **NaN** is a **short form** for- **"Not-a-Number"**.
- NaN is a number which is not **legal number**.
- **Eg:**

```
let notanum = NaN
console.log(notanum); // This will print NaN
console.log(typeof notanum); // This will print number
```

2. **string:** Any **Character** or **grouping of character**, surrounded by **single quotes (' ')** or **double quotes (" ").**

Eg:

```
let str1 = 'Hello'
console.log(str1); // This will print Hello
str1 = "World"
console.log(str1); // This will print World
```

Note: typeof operator

- We can identify type of any value using **typeof** operator.

```
let sillyString = "23"
console.log(sillyString); // This will print 23

// In order to know 23 which is getting printed in console is number or string
console.log(typeof sillyString); // This will print string
```

3. **bigInt**: Any number lying in the **range** of **$-(2^{53}-1)$** to **$(2^{53}-1)$, while declaring bigInt we append 'n' to that number.**

Eg:

```
let bigIntvar = 124676342n
console.log(bigIntvar); // This will print 124676342n
```

4. **boolean:** This datatype **only has two possible values**- either **true** or **false.**

Eg:

```
let boolvar = true
console.log(boolvar); // This will print true
console.log(typeof boolvar); // This will print boolean
boolvar = false
console.log(boolvar); // This will print false
```

5. **null:** This represents the **intentional absence of a value** and is represented by the **keyword null.** It's type is **object.**

Eg:

```
let nullvar = null
console.log(nullvar); // This will print null
console.log(typeof nullvar); // This will print object
```

6. **undefined:** It is also **used to represent the absence of a value** though it has **different use than null**. Undefined means that **the given value does not exists**.

Eg:

```
let undefVar
console.log(undefVar); // This prints undefined because the value to that variable does not exists.
```

7. **symbol:** Symbols are unique identifiers, useful in more complex coding.

Eg:

```
let sym1 = Symbol('123')
console.log(sym1); // This will print Symbol(123)
let sym2 = Symbol('123')
console.log(sym2); // This will print Symbol(123)
console.log(sym1 === sym2); // This will print false
```

Non Primitive Datatypes

1. **arrays** : An Array is an object type designed for storing data collections. It's type is **Object**

Eg:

```
let arr1 = ['Car','Bike','Cycle']

console.log(arr1); // This will print ['Car', 'Bike', 'Cycle']
console.log(typeof arr1); // Object
```

2. **functions:** Functions in JavaScript are **reusable blocks of code** designed to perform specific tasks.

Eg:

```
function greet(){
    console.log("Greetings given");
}
greet()

Output: Greetings given
```

3. **Objects:** An **Object** is a variable **that can hold many variables**. They are collections of **key-value pairs.** It's type is **Object.**

Eg:

```
let car = {
    'brand': "Volkswagon",
    'mileage': "20kmpl",
    'color': "grey",
    'model': 'F50'
}
console.log(car); // This will print {brand: 'Volkswagon', mileage: '20kmpl', color: 'grey', model: 'F50'}
console.log(typeof car); // This will print Object
```