# JavaScript Introduction:

JavaScript can change HTML content.

**Key Characteristics of JavaScript:**

* **Dynamic Typing:** Variables can change their data type during runtime.
* **Object-Oriented:** Everything in JavaScript is an object, including functions.
* **Asynchronous Programming:** Can handle multiple tasks simultaneously without blocking the main thread.
* **Single-Threaded:** Executes code sequentially, but can use asynchronous operations to simulate concurrency.
* **Client-Side Scripting:** Primarily used to add interactivity to web pages.
* **Server-Side Scripting:** Can also be used for server-side development with Node.js.

**Use Cases of JavaScript:**

* **Web Development:**
  + Creating interactive web pages (e.g., animations, form validation, dynamic content)
  + Building web applications (e.g., front-end frameworks like React, Angular, Vue.js)
* **Server-Side Development:**
  + Building web servers and APIs with Node.js
  + Developing command-line tools and automation scripts
* **Game Development:**
  + Creating web-based games and interactive experiences
* **Mobile App Development:**
  + Developing cross-platform mobile apps using frameworks like React Native or Ionic
* **Internet of Things (IoT):**
  + Controlling and interacting with IoT devices

# Data Types Present in JavaScript:

## **Primitive:** Can store only a single value

String, Number, BigInt, Boolean, Undefined, Null, Symbol

|  |
| --- |
| **typeof** "John Doe" // Returns "string"  **typeof** 3.14 // Returns "number"  **typeof** true // Returns "boolean"  **typeof** 234567890123456789012345678901234567890n // Returns bigint  **typeof** undefined // Returns "undefined"  **typeof** null // Returns "object" (kind of a bug in JavaScript)  **typeof** Symbol('symbol') // Returns Symbol |

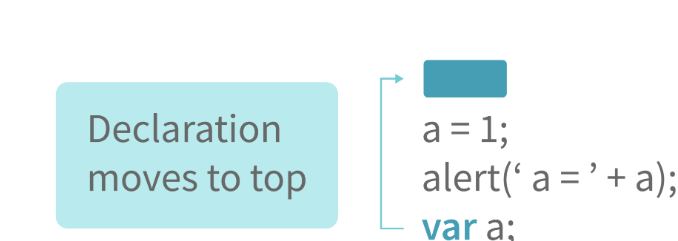
## **Non-Primitive:** used to store multiple and complex values

Object: Used to store collection of data

|  |
| --- |
| // Collection of data in key-value pairs  **var** obj1 = {  x: 43,  y: "Hello world!",  z: **function**(){  **return** this.x;  }  }    // Collection of data as an ordered list    **var** array1 = [5, "Hello", true, 4.1]; |

# **Hoisting:**

Default Variable of JavaScript where All the variables and declarations are moved on top.



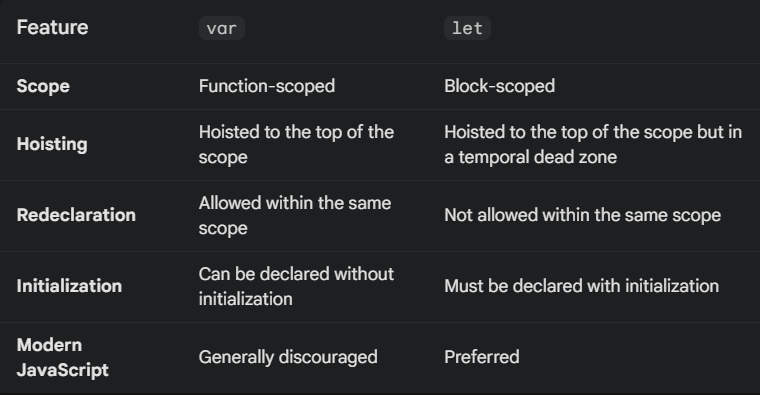
Note: Variable initializations are not hoisted, only variable declarations are hoisted

# Difference between “== “and “=== “

Both are comparison operators. The difference between both the operators is that “==” is used to compare values whereas, “ === “ is used to compare both values and types.

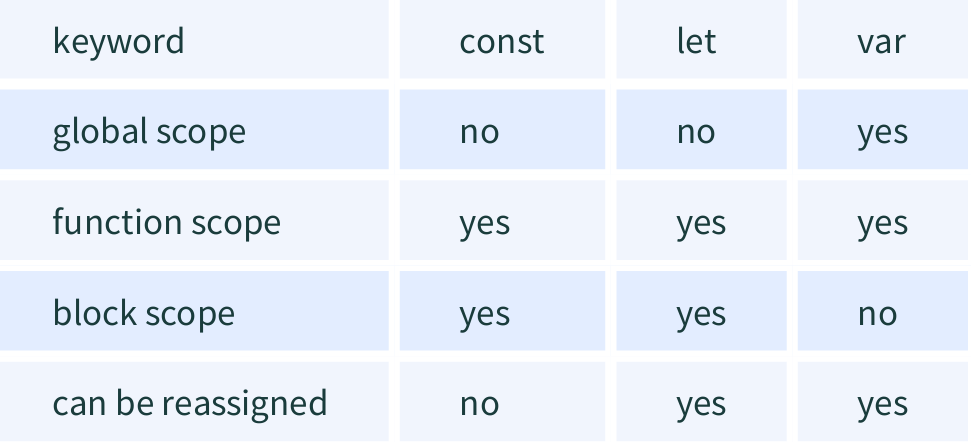
# Var vs let





|  |
| --- |
| function foo() {  var x = 1;  let y = 2;  if (true) {  var x = 3; // Redeclaring x with var is allowed  let y = 4; // Redeclaring y with let is not allowed  console.log(x, y); // Output: 3 4  }  console.log(x, y); // Output: 3 2  }  foo(); |

let prevents accidental redeclarations.



# IMP Concept:

## String Coercion:

When a number is added to

a string, the number type is always converted to the string type.

**Example 1:**

var x = 3;

var y = "3";

x + y // Returns "33"

**Example 2:**

var x = 24;

var y = "Hello";

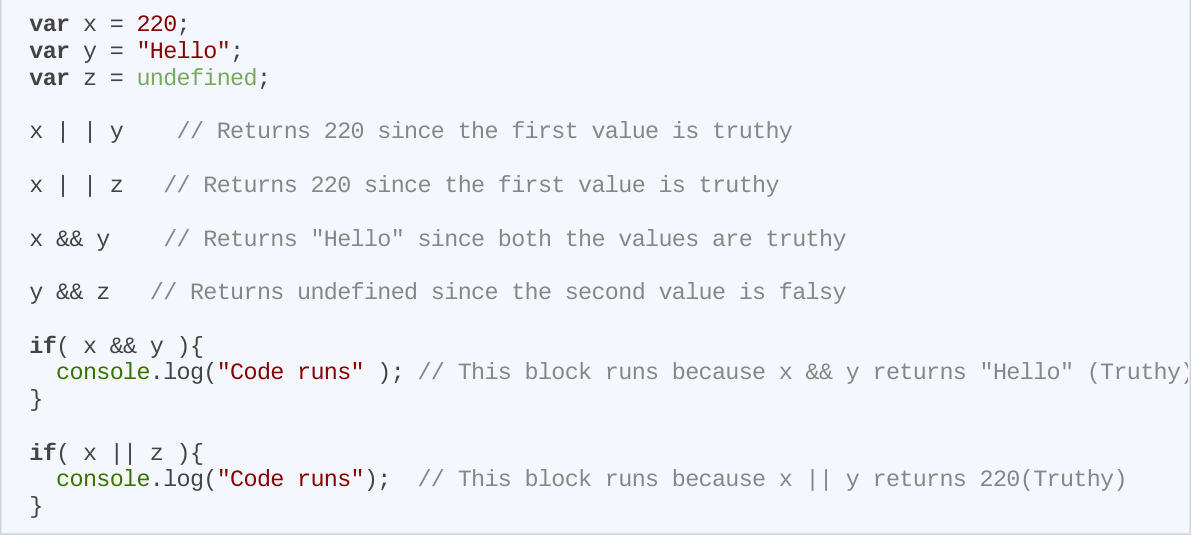
x + y // Returns "24Hello";

+ : type converterts to string

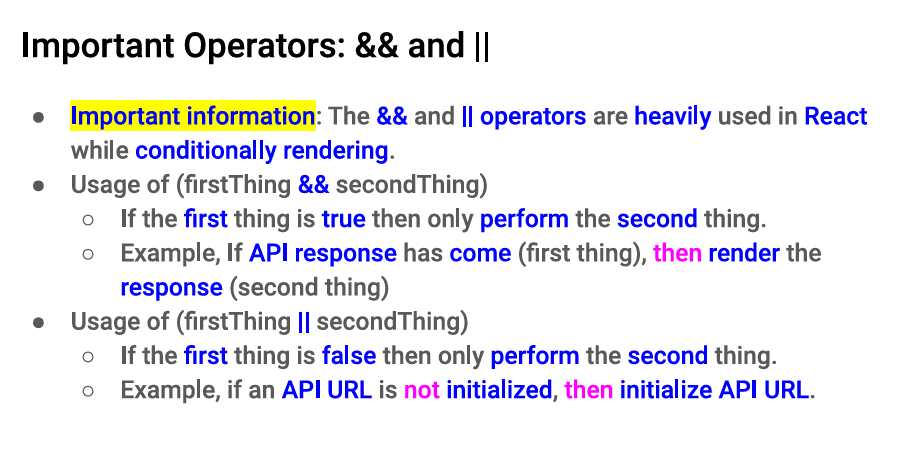
* : type converts to number

|  |
| --- |
| Note:  ‘ + ‘ operator when used to add two numbers, outputs a number. The same ‘ + ‘ operator when used to add two strings, outputs the concatenated string:  Type coercion also takes place when using the ‘ - ‘ operator, but the  difference while using ‘ - ‘ operator is that, a string is converted to a number  and then subtraction takes place |

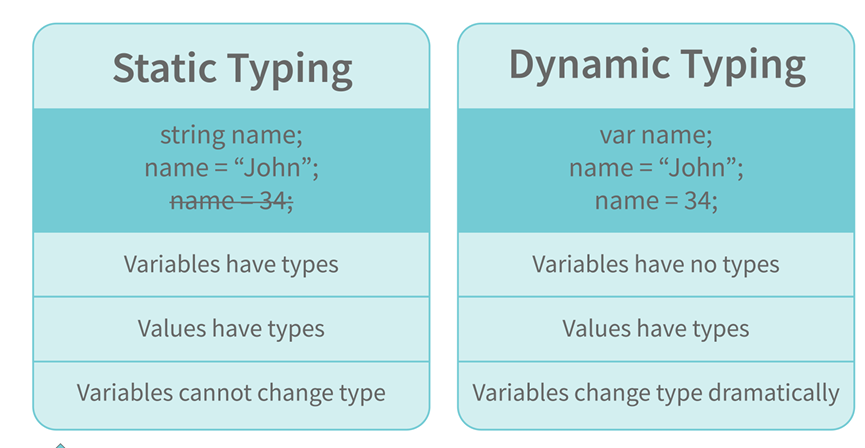
## Logical coericson



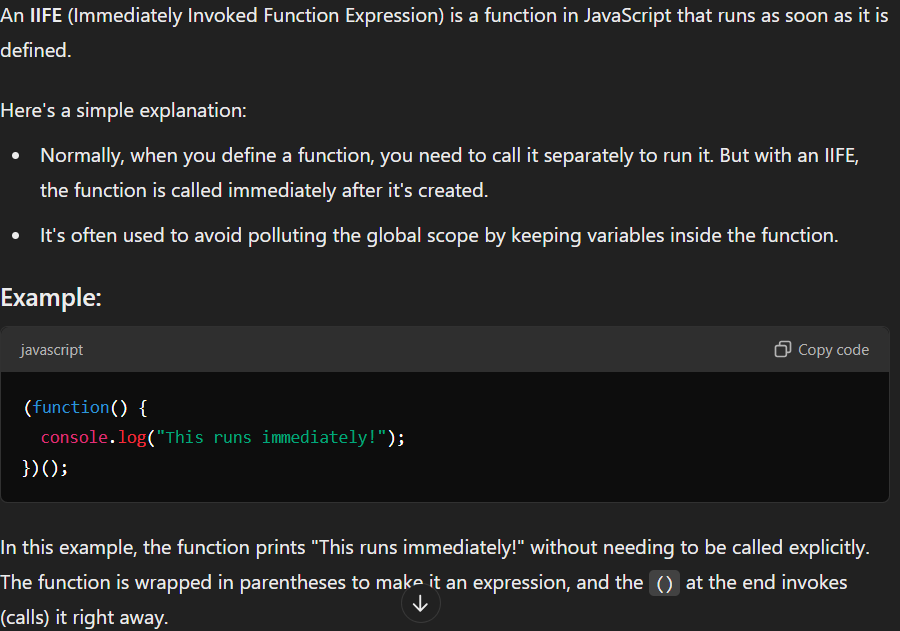
## **&& and || operators:**



# Static vs dynamic:



# **IIFE:(Immediatly invoked functions)**



# for...of vs for…in:

Key Differences:

* What they iterate over: for...of iterates over values, while for...in iterates over keys.
* Use case: for...of is ideal for iterating over arrays and other iterable data structures, while for...in is used for iterating over object properties.
* Inherited properties: for...in includes inherited properties, whereas for...of does not.

Which one to use:

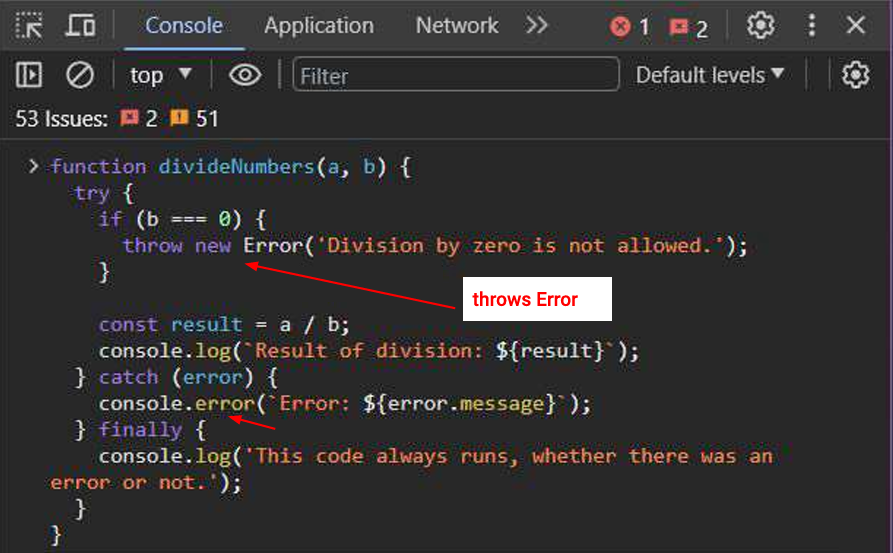
* If you need to access the actual elements of an iterable, use for...of.
* If you need to access the keys of an object, use for...in.

# Try-catch-finally:

Try -> statements to try

Catch -> exception handling statements

Finally -> code that is always executed ,even if there is error or not



# **This keyword:**

**The this keyword in JavaScript refers to the object that is currently executing the code.**

**How it works:**

* **In global scope:** this refers to the global object (usually window in a browser).
* **Inside a function:**
  + **Regular function:** this refers to the object that called the function. If the function is called directly, this refers to the global object.
  + **Method:** this refers to the object that the method belongs to.
  + **Constructor function:** this refers to the newly created object instance.

|  |
| --- |
| **Example:**  JavaScript  let person = {  name: "Alice",  age: 30,  greet: function() {  console.log("Hello, my name is " + this.name);  }};  person.greet(); // Output: Hello, my name is Alice |

**Common use cases:**

* **Creating object methods:** To define methods that operate on the object's properties.
* **Accessing object properties:** To access the object's properties from within its methods.
* **Using this with constructors:** To create new object instances and initialize their properties.

**Remember:** The behavior of this can be influenced by factors like arrow functions, call, apply, and bind. Understanding how these factors affect this is crucial for effective JavaScript programming.

# **Closure:**

This ability of a function to store a variable for further reference even after it is executed is called Closure.

# Prototype:

A prototype is a blueprint of an object. The prototype allows us to use properties and methods on an object even if the properties and methods do not exist on the current object.

# **Callbacks:**

Functions that are used as an argument to another function are called callback functions.

|  |
| --- |
| function divideByHalf(sum){  console.log(Math.floor(sum / 2));  }  function multiplyBy2(sum){  console.log(sum \* 2);  }  function operationOnSum(num1,num2,operation){  var sum = num1 + num2; operation(sum);  }  operationOnSum(3, 3, divideByHalf); // Outputs 3  operationOnSum(5, 5, multiplyBy2); // Outputs 20 |

# Memoization:

Memoization is a form of caching where the return value of a function is cached based on its parameters. If the parameter of that function is not changed, the cached version of the function is returned.

|  |
| --- |
| function addTo256(num){  return num + 256;  }  addTo256(20); // Returns 276  addTo256(40); // Returns 296  addTo256(20); // Returns 276 |

# Constructor function:

Constructor functions are used to create objects in javascript

If we want to create multiple objects having similar properties and methods, constructor functions are used.

|  |
| --- |
| function Person(name,age,gender){  this.name = name;  this.age = age;  this.gender = gender;  }  var person1 = new Person("Vivek", 76, "male");  console.log(person1);  var person2 = new Person("Courtney", 34, "female");  console.log(person2); |

# **charAt()** function is used to retrieve a character from a certain index

Arrow function

Used to declare functions in short way

Arrow functions can only be used as a function expression

# DOM:

**DOM (Document Object Model)** is a programming interface that represents an HTML document as a tree structure of nodes. Each node represents an element, attribute, or text content in the document.

**Key components of the DOM:**

* **Nodes:** The basic building blocks of the DOM. They can be elements, attributes, text nodes, comments, or document nodes.
* **Element nodes:** Represent HTML elements, such as <div>, <p>, and <img>.
* **Attribute nodes:** Represent attributes of elements, such as id, class, and src.
* **Text nodes:** Represent text content within elements.
* **Document node:** Represents the entire HTML document.

**How the DOM works:**

1. **Parsing:** When an HTML document is loaded into a web browser, it is parsed and converted into a DOM tree.
2. **Manipulation:** JavaScript can be used to manipulate the DOM by adding, removing, or modifying nodes.
3. **Rendering:** The browser renders the DOM tree to display the HTML document on the screen.

**Common DOM operations:**

* **Selecting elements:** Using methods like getElementById, getElementsByClassName, and querySelector.
* **Modifying attributes:** Using methods like setAttribute and getAttribute.
* **Creating elements:** Using methods like createElement.
* **Adding and removing elements:** Using methods like appendChild, removeChild, and insertBefore.
* **Handling events:** Using event listeners to respond to user interactions.

**In summary, the DOM provides a way for JavaScript to interact with HTML documents and create dynamic web applications.**

# Spread Operator:

Although the syntax of the spread operator is exactly the same as the rest parameter, the spread operator is used to spreading an array, and object literals. We also use spread operators where one or more arguments are expected in a function call.

# Promise:

* Promises are used to handle asynchronous operations in JavaScript.
* Promise object has four states
* Before promises, callbacks were used to handle asynchronous operations. But due to
* the limited functionality of callbacks, using multiple callbacks to handle asynchronous code can lead to unmanageable code.

1. **Pending** - Initial state of promise. This state represents that the promise has neither been fulfilled nor been rejected, it is in the pending state.
2. **Fulfilled** - This state represents that the promise has been fulfilled, meaning the async operation is completed.
3. **Rejected** - This state represents that the promise has been rejected for some reason, meaning the async operation has failed.
4. **Settled** - This state represents that the promise has been either rejected or fulfilled.

A promise is created using the Promise constructor which takes in a callback function with two parameters, resolve and reject respectively.

**resolve** is a function that will be called when the async operation has been successfully completed.

**reject** is a function that will be called, when the async operation fails or if some error occurs