Data Types and Operators

Shoaib Ghachi

MCT | Technical Trainer | Mentor |SME

Overview

Data Types and Operator.

- -Primitive Data Types
- -Complex Data Types
- -var, let and const keywords
- -Operators
  - -Arithmatic Operators
  - -Comparison Operator
  - -Assignment Operator
  - -Logical Operator

Data Types

- In JavaScript, data types are classifications that define the type of values that can be assigned to variables. JavaScript has several built-in data types, including:
- **Primitive Data Types:**
  - **String:** Represents a sequence of characters, enclosed in single or double quotes.

    e.g.
    ```
    let name = "John Doe";
    let message = 'Hello, world!';
    ```

  - **Number:** Represents numeric values, including integers and floating-point numbers.

    e.g.
    ```
    let age = 25;
    let temperature = 98.6;
    ```

- **Boolean:** Represents either the value true or false.

  e.g.
  ```
  let isStudent = true;
  let hasCar = false;
  ```

- **Undefined:** Represents a variable that has been declared but has not been assigned a value.

  e.g.
  ```
  let myVar;
  console.log(myVar); // Output: undefined
  ```

- **Null:** Represents the intentional absence of any object value.

  e.g.
  ```
  let myVariable = null;
  ```

- **Complex Data Types:**
  - **Object**: Represents a collection of key-value pairs, where values can be of any data type. Objects can also have methods, which are functions associated with the object.

```
// Example of a complex data type (Object)
let person = {
  name: "John Doe",
  age: 30,
  isStudent: true,
  address: {
    city: "New York",
    zipCode: "10001"
  },
  hobbies: ["Reading", "Hiking", "Cooking"],
  sayHello: function() {
    return "Hello, my name is " + this.name + ".";
  }
};
```

```
// Accessing object properties
console.log(person.name); // Output: "John Doe"
console.log(person.age); // Output: 30
console.log(person.address.city); // Output: "New York"
console.log(person.hobbies[0]); // Output: "Reading"

// Calling the function within the object
console.log(person.sayHello()); // Output: "Hello, my name
is John Doe."
```

var, let and const Keywords

- **var**: Variables declared with **var are function-scoped**, which means they are accessible within the function in which they are declared. If declared outside any function, they become global variables and are accessible throughout the entire script. They are also hoisted to the top of their scope.

```
// Example of 'var'
function exampleVar() {
if (true) {
    var x = 10;
    console.log(x); // Output: 10
  }
  console.log(x); // Output: 10 (var has function scope)
}
exampleVar();
```

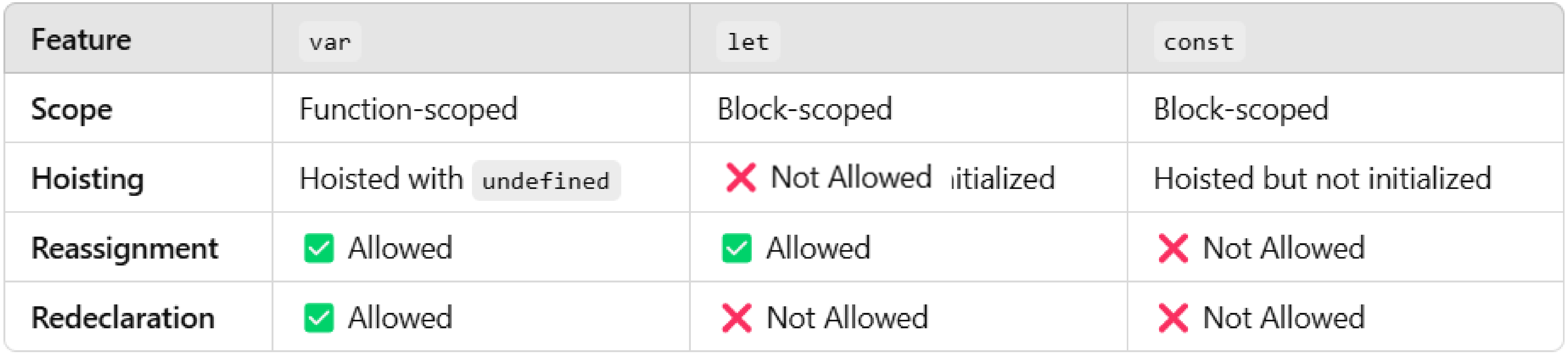
- **let**: Variables declared with let are block-scoped, meaning they are accessible within the block (statements enclosed in curly braces) in which they are defined. They are not hoisted to the top of their scope, and attempting to access them outside the block will result in an error.

```
// Example of 'let'
function exampleLet() {
  if (true) {
    let y = 20;
    console.log(y); // Output: 20
  }
  // console.log(y); // Error: 'y' is not defined (let has block scope)
}
exampleLet();
// console.log(y); // Error: 'y' is not defined (y is not accessible outside the function)
```

- **const**: Variables declared with const are also block-scoped like let, but they cannot be reassigned after their initial assignment. They are read-only, making them useful for declaring constants or values that should not be changed.

```
// Example of 'const'
function exampleConst() {
  const z = 30;
  console.log(z); // Output: 30
  // z = 40; // Error: Assignment to constant variable (const variables are read-only)
}
exampleConst();
// console.log(z); // Error: 'z' is not defined (z is not accessible outside the function)
```

In modern JavaScript, it's generally recommended to use let and const instead of var due to the improved scoping rules and the prevention of accidental reassignments in the case of const.



Feature	`var`	`let`	`const`
Scope	Function-scoped	Block-scoped	Block-scoped
Hoisting	Hoisted with `undefined`	❌ Not Allowed itialized	Hoisted but not initialized
Reassignment	✅ Allowed	✅ Allowed	❌ Not Allowed
Redeclaration	✅ Allowed	❌ Not Allowed	❌ Not Allowed

How Hoisting Works?

- **Variables (var, let, const):** var is hoisted but initialized as undefined. let and const are hoisted, but they remain uninitialized (stay in the Temporal Dead Zone until assigned).
- **Functions (function):** Function declarations are hoisted with their definitions, so you can call them before defining them. Function expressions are not hoisted.

Examples of Hoisting:

1️⃣ Hoisting with var :

```
console.log(a); // ✅ `undefined` (Hoisted but not assigned)

var a = 10;

console.log(a); // 10
```

Behind the scenes, JavaScript interprets it as:

```
var a;  // Declaration is hoisted to the top

console.log(a); // undefined

a = 10; // Assignment remains in place

console.log(a); // 10
```

2️⃣ Hoisting with let and const

```
console.log(b); // ❌ ReferenceError: Cannot access 'b' before initialization

let b = 20;

console.log(b);
```

In addition to data types, JavaScript also provides various operators that allow you to perform operations on values. Here are some of the commonly used operators in JavaScript:

- **Arithmetic Operators:**

	let num1 = 10;
	let num2 = 5;
+ (Addition)	let sum = num1 + num2; // 10 + 5 = 15
- (Subtraction)	let difference = num1 - num2; // 10 - 5 = 5
* (Multiplication)	let product = num1 * num2; // 10 * 5 = 50
/ (Division)	let quotient = num1 / num2; // 10 / 5 = 2
% (Modulus)	let remainder = num1 % num2; // 10 % 5 = 0
++ (Increment)	let incrementNum = 7; incrementNum++; //8
-- (Decrement)	let decrementNum = 9; decrementNum--; //8

- **Comparison Operators:**

	let num1 = 10;
	let num2 = 5;
== (Equality)	console.log(num1 == num2); // false
===(Strict Equality)	console.log(5 === 5); // Output: true (both operands are numbers and have the same value)
!= (Inequality)	console.log(num1 !== num2); // true
> (Greater than)	console.log(num1 > num2); // true
< (Less than)	console.log(num1 < num2); // false
>= (Greater than or equal to)	console.log(num1 >= num2); // true
<= (Less than or equal to)	console.log(num1 <= num2); // false

Assignment Operators:

= (Assignment)	let num = 10; // Assign the value 10 to the variable 'num’
+= (Addition assignment)	num += 5; // Equivalent to: num = num + 5; //15
-= (Subtraction assignment)	num -= 5; // Equivalent to: num = num - 5; //10
*= (Multiplication assignment)	num *= 3; // Equivalent to: num = num * 3;//30
/= (Division assignment)	num /= 5; // Equivalent to: num = num / 6;
%= (Modulus assignment)	num %= 3; // Equivalent to: num = num % 5; //0

- **Logical Operators:**

&& (Logical AND)

```
let x = 5;
let y = 10;

// Both x and y are greater than 0, so the result is true
let result1 = x > 0 && y > 0;
console.log(result1); // Output: true
```

|| (Logical OR)

```
let x = 5;
let y = 10;

// At least one of x or y is greater than 10, so the result is true
let result1 = x > 10 || y > 10;
console.log(result1); // Output: true
```

! (Logical NOT)

```
let isStudent = true;


// The expression isStudent is true, but !isStudent negates it to false
let result1 = !isStudent;
console.log(result1); // Output: false
```

- **String Operators:** + (String concatenation)

```
let firstName = "John";
let lastName = "Doe";


// Concatenating the two strings using the + operator
let fullName = firstName + " " + lastName;
console.log(fullName); // Output: "John Doe"
```

- **Conditional (Ternary) Operator:**

condition ? expression1 : expression2

```
let age = 25;
// Using the ternary operator to check if age is greater than or equal to 18
// If true, assign "Adult" to status, otherwise, assign "Minor"
let status = age >= 18 ? "Adult" : "Minor";
console.log(status); // Output: "Adult"
```

- **Typeof Operator:** typeof (Returns the data type of a value)

```
let num = 42;
let str = "Hello, world!";
let arr = [1, 2, 3];
let obj = { name: "John", age: 30 };
let isTrue = true;
let func = function() {};
```

```
console.log(typeof num); // Output: "number"
console.log(typeof str); // Output: "string"
console.log(typeof arr); // Output: "object"
console.log(typeof obj); // Output: "object"
console.log(typeof isTrue); // Output: "boolean"
console.log(typeof func); // Output: "function“
```

References

https://www.w3schools.com/js/js_datatypes.asp

https://www.programiz.com/javascript/data-types

https://www.javatpoint.com/javascript-data-types

https://www.geeksforgeeks.org/variables-datatypes-javascript/

https://www.tutorialrepublic.com/javascript-tutorial/javascript-data-types.php

https://www.w3schools.com/js/js_operators.asp

https://www.javatpoint.com/javascript-operators