

1. Explain how var works in JavaScript. What is variable hoisting? Give a code example.

>> **var:** is used to declare variables in global scope, and function scope but not in block scope

>> **Hoisting:** is moving variables and function declarations to the top of their containing scope (global or function scope) during the compilation phase, Then Assigning their values in code respectively.

```
console.log(a);  
// Output: undefined (But Not Error Because of Hoisting)  
  
var a = 5;  
  
console.log(a);  
// Output: 5
```

2. What is the scope of a variable declared with var inside a function? What about inside a block (e.g., an if statement)?

>> **In A Function:** It's Function Scope (Can't Access it outside the Function).

>> **In A Block:** It's Global Scope (Can Access It From Out Side The Block).

3- List all JavaScript primitive types in ES5. Give an example of each.

```
// Undefined

var x;
console.log(x); // Output: undefined
// -----
// Null

var y = null;
console.log(y); // Output: null
// -----
// Boolean

var z = true;
console.log(z); // Output: true
// -----
// Number

var num = 25;
console.log(num); // Output: 25
// -----
// String

var message = "Hello, World!";
console.log(message); // Output: Hello, World!
// -----

var obj = { name: "Abdelrahman" };
console.log(obj.name); // Output: Abdelrahman
// -----
```

4- What is the difference between a primitive type and an object type? Give an example where this difference is important.

>> **Primitive Type:** The Variable Can Be Assigned To A Single Value
(Number, String, Boolean, ...)

>> **Object Type:** The Variable Can Have Multiple Attributes & Methods which are related or can be used together.

```
var person = {
  firstName: "Abdelrahman",
  lastName: "Salem",
  age: "25",
  skills: ["HTML", "CSS", "JavaScript", "Python", "Database"],
  fullName() {
    return `${this.firstName} ${this.lastName}`;
  },
};

console.log(person.fullName());
// Output: Abdelrahman Salem
```

5. Create a number, string, and boolean using both literal and constructor syntax. Show the difference in their types using `typeof`.

```
let numLiteral = 42;
let numObject = new Number(42);

let strLiteral = "Hello";
let strObject = new String("Hello");

let boolLiteral = true;
let boolObject = new Boolean(true);

console.log(typeof numLiteral); // "number"
console.log(typeof numObject); // "object"

console.log(typeof strLiteral); // "string"
console.log(typeof strObject); // "object"

console.log(typeof boolLiteral); // "boolean"
console.log(typeof boolObject); // "object"
```

6. Why is it generally recommended to use literals instead of constructors for primitive types?

- Literals Are Easier And Faster Than Constructors As No Object Creation is Need & You Can Access The Variable Directly without Pointers or Addresses

7. Given the following code, what will be the output? Explain why.

```
var x = 123.4567;  
console.log(x.toFixed(2));  
console.log(x.toPrecision(4));
```

```
var x = 123.4567;  
console.log(x.toFixed(2)); // 123.46  
console.log(x.toPrecision(4)); // 123.5
```

8. What is NaN? How can you check if a value is NaN? Give an example.

- represents the result of an invalid or undefined mathematical operation, such as dividing zero by zero or parsing a non-numeric string as a number.

```
var result = 4 - "abc"; // This will be NaN  
  
console.log(result); // Output: NaN  
  
console.log(isNaN(result)); // true  
console.log(Number.isNaN(result)); // true
```

9. What is the difference between parseInt, parseFloat, and Number? Give an example for each.

- **parseInt**: Converts a string to an integer (Return NaN if can't be converted)
- **parseFloat**: Converts a string to a floating-point number (Return NaN if can't be converted)
- **Number**: A constructor function that converts a value to a number. (Return NaN if can't be converted)

```
// parseInt() examples
console.log(parseInt("123")); // 123 (valid)
console.log(parseInt("123abc")); // 123 (valid, stops at non-digit)
console.log(parseInt("abc123")); // NaN (invalid)
console.log(parseInt("12.34")); // 12 (valid, stops at decimal)
console.log(parseInt("")); // NaN (invalid)

// parseFloat() examples
console.log(parseFloat("123.45")); // 123.45 (valid)
console.log(parseFloat("123.45abc")); // 123.45 (valid, stops at non-digit)
console.log(parseFloat("abc123.45")); // NaN (invalid)
console.log(parseFloat("")); // NaN (invalid)
console.log(parseFloat("12")); // 12 (valid)

// Number() examples
console.log(Number("123")); // 123 (valid)
console.log(Number("123.45")); // 123.45 (valid)
console.log(Number("123abc")); // NaN (invalid)
console.log(Number("")); // 0 (valid, empty string is 0)
console.log(Number("abc")); // NaN (invalid)
```

10. What is the difference between implicit and explicit type casting? Give an example of each.

- **Implicit** type casting occurs when JavaScript automatically converts one data type to another
- **Explicit** type casting occurs when the programmer deliberately converts a value from one type to another

```
// Implicit Type Casting (Type Coercion)
let result1 = "5" + 2; // JavaScript converts 2 to '2', result is '52'

// Explicit Type Casting
let result2 = Number("5") + 2; // '5' is explicitly converted to 5, result is 7
```

11. What will be the result and type of the following expressions? Explain your answer.

- true + 5 - "10" - 2 - 12 - "1a" - 5 / 0 - 5 + undefined

```
console.log(true + 5); // Output: 6
// Explanation: true is casted to 1

console.log("10" - 2); // Output: 8
// Explanation: "10" is casted to 10 (number)

console.log(12 - "1a"); // Output: NaN
// Explanation: "1a" cannot be converted to a number

console.log(5 / 0); // Output: Infinity
// Explanation: Division by zero in JavaScript returns Infinity

console.log(5 + undefined); // Output: NaN
// Explanation: undefined is casted to NaN
```

12. What will be logged to the console in the following code? Explain each step.

```
var a = "15.5";  
var b = +a;  
console.log(b, typeof b);
```

```
var a = "15.5";  
var b = +a;  
  
console.log(b, typeof b);  
// Output: 15.5  number  
// The unary plus + operator performs explicit type conversion to a number
```

13. What will be the output, Explain why.

```
var result = 20 > true < 5 == 1;  
  
console.log(result);  
// Output: true  
  
// Explanation:  
// 1. 20 > true => 20 > 1 => true  
// 2. 1 < 5 => true  
// 3. 1 == 1 => true
```


14. Write a function that takes a string and returns true if it can be converted to a valid number, and false otherwise.

```
function validNumer(s) {
  if (typeof s !== "string" && typeof s !== "number") return false;
  if (typeof s === "string") {
    s = s.trim();
    if (s.length === 0) return false;
  }

  var num = Number(s);
  return !isNaN(num) && isFinite(num);
}

console.log(validNumer("123")); // true
console.log(validNumer(" 45.67 ")); // true
console.log(validNumer("abc")); // false
console.log(validNumer("")); // false
console.log(validNumer("12e3")); // true
console.log(validNumer("12e3.4")); // false
console.log(validNumer("Infinity")); // false
console.log(validNumer(NaN)); // false
console.log(validNumer(456)); // true
console.log(validNumer(null)); // false
console.log(validNumer(undefined)); // false
```

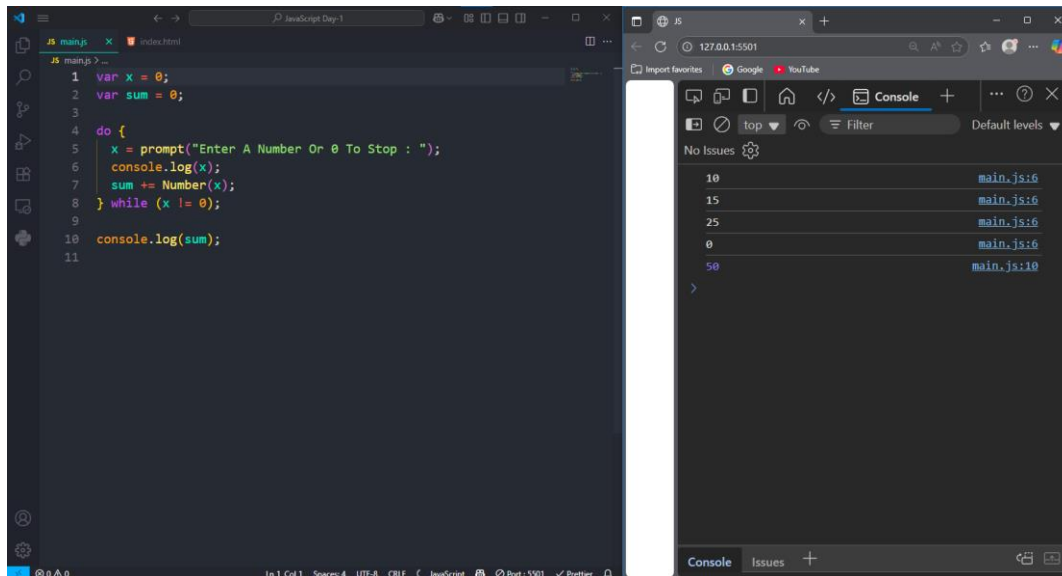
15. Write a program that prints all numbers from 1 to 20 using a while loop.

```
function print20() {
  var n = 1;

  while (n <= 20) {
    console.log(n);
    n++;
  }
}

print20();
```

16. Write a program that asks the user to enter numbers until they enter 0, using a do...while loop. After the loop ends, print the sum of all entered numbers (excluding 0).



The screenshot shows a code editor on the left with the following JavaScript code:

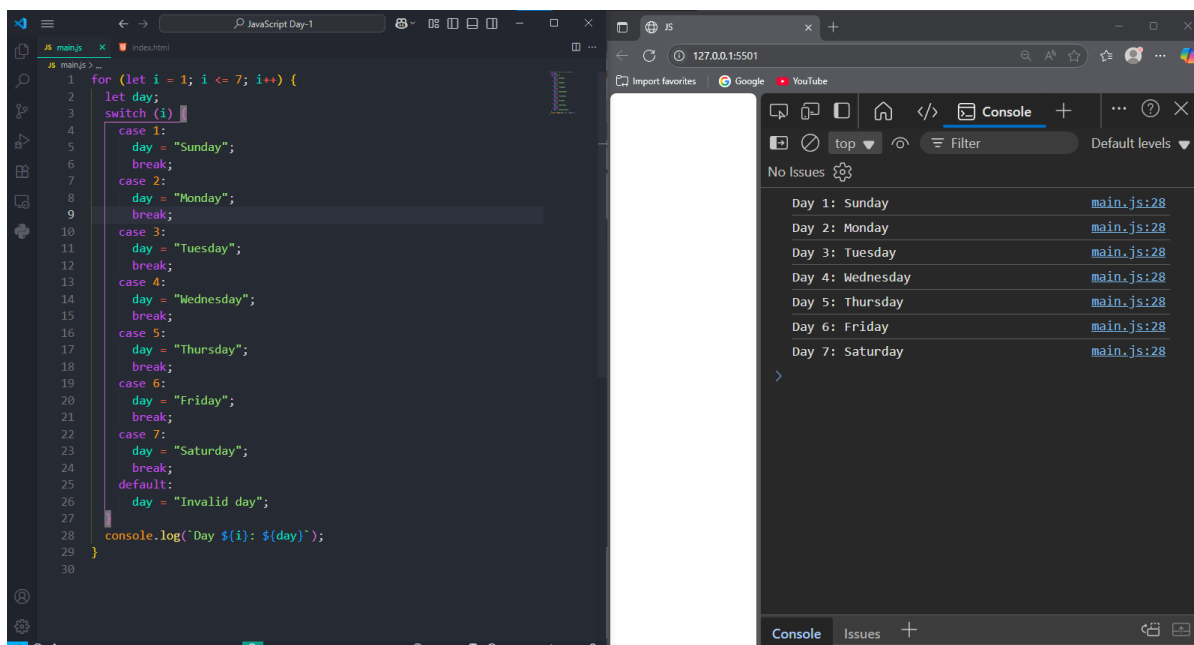
```
1 var x = 0;
2 var sum = 0;
3
4 do {
5   x = prompt("Enter A Number Or 0 To Stop : ");
6   console.log(x);
7   sum += Number(x);
8 } while (x != 0);
9
10 console.log(sum);
11
```

The browser console on the right shows the output of the program:

```
10
15
25
0
50
```

The console also shows the file path `main.js:6` for the first three outputs and `main.js:10` for the final sum output.

17. Write a program that takes a number from 1 to 7 and prints the corresponding day of the week using a switch statement. Use a for loop to test your program with all numbers from 1 to 7.



The screenshot shows a code editor on the left with the following JavaScript code:

```
1 for (let i = 1; i <= 7; i++) {
2   let day;
3   switch (i) {
4     case 1:
5       day = "Sunday";
6       break;
7     case 2:
8       day = "Monday";
9       break;
10    case 3:
11      day = "Tuesday";
12      break;
13    case 4:
14      day = "Wednesday";
15      break;
16    case 5:
17      day = "Thursday";
18      break;
19    case 6:
20      day = "Friday";
21      break;
22    case 7:
23      day = "Saturday";
24      break;
25    default:
26      day = "Invalid day";
27  }
28  console.log("Day ${i}: ${day}");
29 }
30
```

The browser console on the right shows the output of the program:

```
Day 1: Sunday
Day 2: Monday
Day 3: Tuesday
Day 4: Wednesday
Day 5: Thursday
Day 6: Friday
Day 7: Saturday
```

The console also shows the file path `main.js:28` for each output line.