

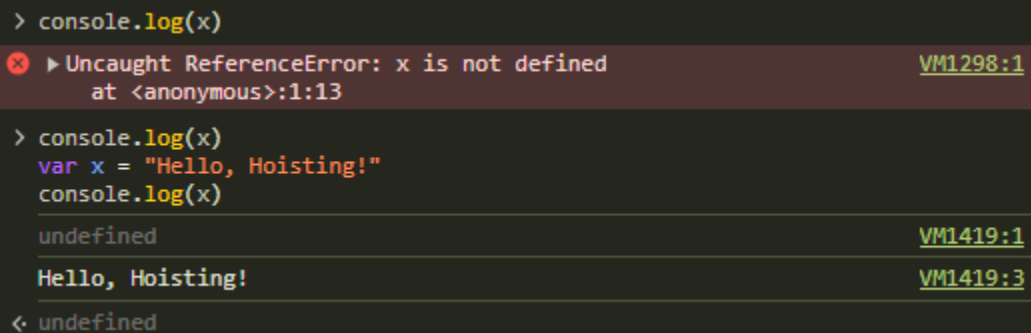
JavaScript Assignment – Lecture 1

Instructions: Answer all questions. Write code where required and explain your answers clearly.

Part 1: Variables and Scope

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

- In JavaScript, var is a keyword used to declare variables. Variables declared with var have function scope, meaning they are accessible throughout the entire function in which they are declared, or globally if declared outside of any function.
- Hoisting is a JavaScript mechanism where variable and function declarations are moved to the top of their containing scope during the compilation phase, before the code is executed.



```
> console.log(x)
VM1298:1 Uncaught ReferenceError: x is not defined
    at <anonymous>:1:13

> console.log(x)
var x = "Hello, Hoisting!"
console.log(x)
undefined
Hello, Hoisting!
< undefined
```

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

- If you use **var** inside a function, the variable lives in the whole function.
- If you use **var** inside a block (like an if), it still lives in the whole function — not just that block.

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

- **Number** → var x = 42;
- **String** → var name = "Alice";
- **Boolean** → var isOn = true;
- **Undefined** → var y; // undefined
- **Null** → var z = null;

```
> var x = 42
< undefined
> typeof(x)
< 'number'
> var name= 'Hima'
  typeof(name)
< 'string'
> var isOn= true;
  typeof(isOn)
< 'boolean'
> var y
  typeof(y)
< 'undefined'
> var z = null;
  typeof(z)
< 'object'
```

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

- Primitives are simple values (like numbers or strings) and are compared **by value**. Objects are collections (like arrays or objects) and are compared **by reference**.

```
> var a = 5
var b = 5
console.log(a==b)
true VM67:3
< undefined
> x= new Number(25)
y= new Number(25)
console.log(x==y)
false VM77:3
< undefined
> console.log(x.valueOf == y.valueOf)
true VM125:1
< undefined
> |
```

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

```
> var x = 42
var y= new Number(33)
console.log(typeof(x))
console.log(typeof(y))
number VM526:3
object VM526:4
< undefined
> var x = true
var y= new Boolean(true)
console.log(typeof(x))
console.log(typeof(y))
boolean VM561:3
object VM561:4
< undefined
> var x = "Hello"
var y= new String("it's me ")
console.log(typeof(x))
console.log(typeof(y))
string VM599:3
object VM599:4
< undefined
> |
```

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

- **Easier** → `var x = 5;` is simpler than `var x = new Number(5);`
- **Faster** → literals are primitives, constructors make objects (heavier).

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));
```

- `toFixed(n)` → focuses on decimals (after the dot).
- `toPrecision(n)` → focuses on total important digits (before + after the dot).

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

123.46
123.5
```

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

- **NaN** = “Not-a-Number”, result of invalid math.
- `isNaN(x)` → loose check (does type conversion).
- `Number.isNaN(x)` → strict check (recommended).

```
> var x = 0/0
  console.log(Number.isNaN(x));

true
```

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

- `parseInt` → converts to integer (drops decimals).
- `parseFloat` → converts to float (keeps decimals).
- `Number` → strict conversion, whole string must be numeric.

```
> console.log(parseInt(123.45));
  console.log(parseFloat(123.45));
  console.log(Number("123.45"));
  console.log(Number("123px"));

123
123.45
123.45
NaN
```

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

- Implicit casting → JS converts type automatically.
- Explicit casting → you convert type manually.

```
> var a = "5" + 2;  
  console.log(a);  
var b = Number("5") + 2;  
  console.log(b)  
52  
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);  
  console.log("10"-2);  
  console.log(12 - "1a");  
  console.log(5/0);  
  console.log(5+undefined);  
6  
8  
NaN  
Infinity  
NaN
```

- true + 5 → 6 (number)
 - o true becomes 1.
- "10" - 2 → 8 (number)
 - o "10" becomes number 10.
- 12 - "1a" → NaN (number)
 - o "1a" can't be converted.

- $5 / 0 \rightarrow \text{Infinity}$ (number)
 - o Division by zero gives Infinity.
- $5 + \text{undefined} \rightarrow \text{NaN}$ (number)
 - o undefined becomes 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);`

- a is the string "15.5".
- +a (unary plus) converts the string to a number.
- So b becomes 15.5 (type number).

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

13. What will be the output of:

`var result = 20 > true < 5 == 1;`

`console.log(result);`

Explain why.

- $0 > \text{true} \rightarrow \text{true}$ (true = 1).
- $\text{true} < 5 \rightarrow 1 < 5 \rightarrow \text{true}$.
- $\text{true} == 1 \rightarrow \text{true}$.

```
> var result= 20> true < 5 == 1 ;
console.log(result)
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 isValidNumber(str) {  
    return !isNaN(str);  
}  
← undefined  
> console.log(isValidNumber("123"));  
console.log(isValidNumber("abc"));  
  
true  
  
false
```

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

```
> var i =1 ;  
← undefined  
> while (i<=20){  
    console.log(i++);  
}  
1 VM2954:2  
2 VM2954:2  
3 VM2954:2  
4 VM2954:2  
5 VM2954:2  
6 VM2954:2  
7 VM2954:2  
8 VM2954:2  
9 VM2954:2  
10 VM2954:2  
11 VM2954:2  
12 VM2954:2  
13 VM2954:2  
14 VM2954:2  
15 VM2954:2  
16 VM2954:2  
17 VM2954:2  
18 VM2954:2  
19 VM2954:2  
20 VM2954:2  
← undefined
```

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).

```
> var sum=0;
var num;
do {
    num=prompt("Enter a number ( 0 to stop):");
    num=Number(num);
    sum=sum+num;
} while (num != 0);
console.log("Sum = " + sum)

Sum = 20
```

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.

```
> for ( var i = 1 ; i <=7 ; i++){
    switch(i){
        case 1:console.log("Sunday");break;
        case 2:console.log("Monday");break;
        case 3:console.log("Tuesday");break;
        case 4:console.log("Wednesday");break;
        case 5:console.log("Thursday");break;
        case 6:console.log("Friday");break;
        case 7:console.log("Saturday");break;
    }
}

Sunday VM3657:3
Monday VM3657:4
Tuesday VM3657:5
Wednesday VM3657:6
Thursday VM3657:7
Friday VM3657:8
Saturday VM3657:9
< undefined
```