

# Let's Learn JS Fundamentals

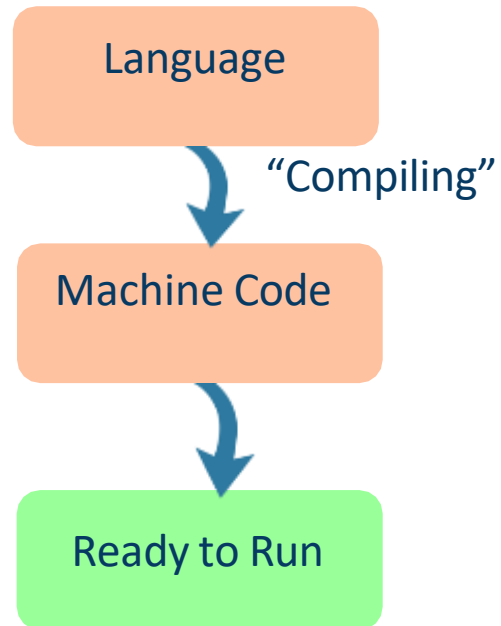
# Agenda – Part 1 (Basics)

- Learn variables & Data types
- Primitive vs Non primitive data types
- Null vs Undefined
- var vs let vs const
- Re-declaration and Re-assignment
- Scoping
- Hoisting

# Compiled vs Interpreted language

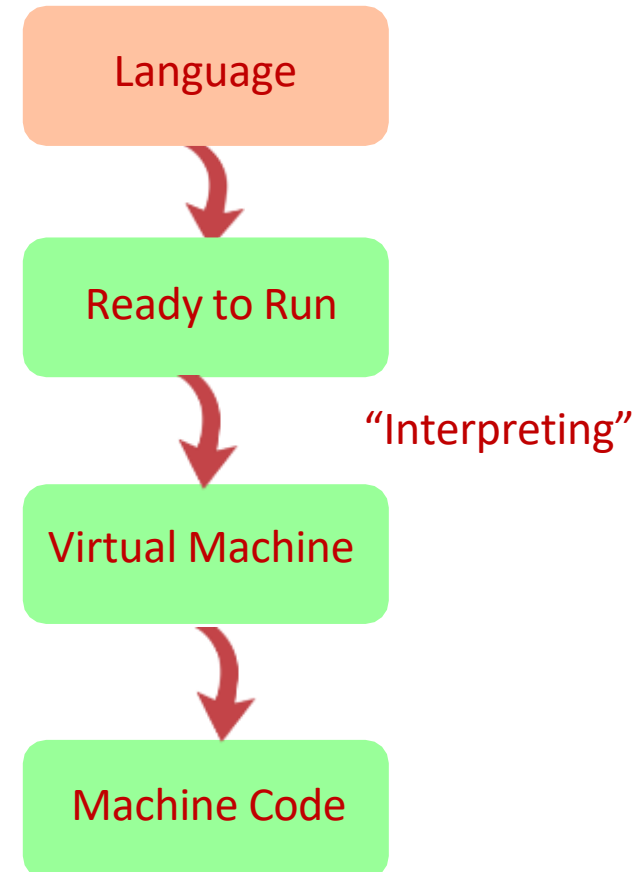
Compiled Language

C, C++, Fortran, Pascal, Java

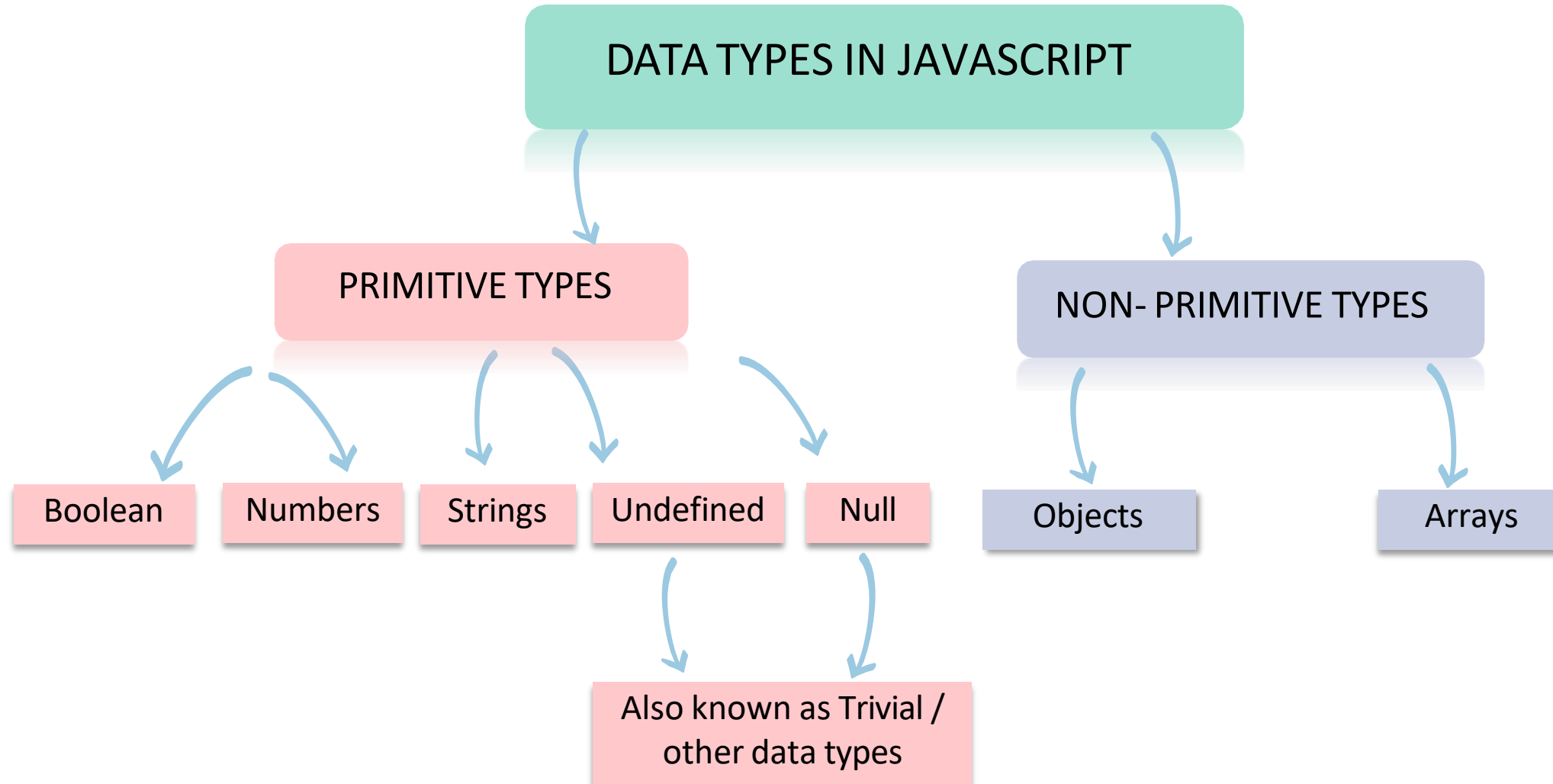


Interpreted Language

Python, PHP, Ruby, JavaScript



# Understanding Data Types



# Null & Undefined in JS

*Null*



*Undefined*

# About variable declaration

- ❑ Variable declaration in JavaScript refers to the process of creating a named storage space in memory, which can hold a value.
- ❑ When a variable is declared, JavaScript allocates a memory location for that variable and initializes it with a default value (e.g., undefined for uninitialized variables).
- ❑ Example

```
var courseName = "Playwright"
```

- var → variable keywords / variable declaration
- courseName → variable name
- = → assignment operator
- “ ” → Used to declare a string.
- Playwright → Value of the assigned to the variable.

# About “typeof” operator

The **typeof** operator in JavaScript is used to determine the data type of a value or variable. It returns a string that indicates the type of the operand.

Example :

```
1  var courseName = "Playwright"
2  console.log("The datatype of courseName is" + typeof courseName); // String
3
4  var latestVersion = 1.49
5  console.log("The datatype of courseName is" + typeof latestVersion); // Number
6
7  var PlaywrightTestAutomationTool = true;
8  console.log("The datatype of courseName is" + typeof PlaywrightTestAutomationTool); // Boolean
9
10 var accountNumber;
11 console.log("The datatype of accountNumber is" +accountNumber); // Undefined
12
13 var landlineNumber =null;
14 console.log(landline);
15 console.log("The datatype of landlineNumber is" +landlineNumber); // Object
```

# Classroom on typeof operator :

## Assignment Details:

Declare variables using `let` for different data types in JavaScript, and verify their types using `typeof`, including an uninitialized variable.

## Assignment Requirements:

Create the following variables using let (not using var) and check their typeOf

- a) firstName
- b) companyName
- c) mobileNumber
- d) isAutomation
- e) hasPlaywright (do not assign)

Print and confirm the values and data types

## Hints to Solve:

Focus on initializing variables with different values, including a string, number, boolean, and leave one variable undefined to practice with `typeof`.

## Expected Outcome:

Upon completion, you should be able to:

- Grasp the concepts of different data types in JavaScript



## *Difference between*

*var*      *vs*      *let*      *vs*      *const*

*In terms of Re-declaration, Re-assignment, Scoping and Hoisting*



# Behavior of “var” keyword:

## Redeclaration using var :

Variables declared with var can be redeclared within the same scope without throwing an error.

```
1  var x = 10;  
2  var x = 20; // Allowed  
3  console.log(x); // 20
```

## Reassignment using var :

Variables declared with var can be reassigned to a new value at any time.

```
1  var y = 30;  
2  y = 40; // Allowed  
3  console.log(y); // 40
```

# Behavior of “let” keyword:

## Redeclaration in let :

Variables declared with let cannot be redeclared in the same scope.

```
1  let x = 10;  
2  let x = 20; // ✗ Error: Identifier 'x' has already been declared
```

## Reassignment in let :

Variables declared with let can be reassigned new values.

```
1  let y = 30;  
2  y = 40; // ✓ Allowed  
3  console.log(y); // 40  
4
```

# Behavior of “const” keyword:

## Redeclaration in const :

Variables declared with const cannot be redeclared in the same scope.

```
1  const x = 10;  
2  const x = 20; // ✗ Error: Identifier 'x' has already been declared
```

## Reassignment in const :

Variables declared with const cannot be reassigned after their initial assignment.

```
1  const y = 30;  
2  y = 40; // ✗ Error: Assignment to constant variable
```

# Hoisting

- Hoisting is a simple mechanism where the JavaScript interpreter moves the variable and function declarations to the top of the code block

● `console.log(x);` // Outputs: undefined  
`var x = 10;`  
`console.log(x);` // Outputs: 10

## Hoisting in var :

Variables declared with var are hoisted to the top of their function or global scope, but they are initialized as undefined

```
1 console.log(a); // undefined (Hoisted, but not initialized)
2 var a = 100;
```

# Hoisting

## Hoisting in let :

Variables declared with let are hoisted, but they are not initialized. Accessing them before their declaration results in a ReferenceError.

```
1 console.log(a); // ✗ Error: Cannot access 'a' before initialization
2 let a = 100;
```

## Hoisting in const :

Variables declared with const are hoisted, but they are not initialized. Accessing them before their declaration results in a ReferenceError.

```
1 console.log(a); // ✗ Error: Cannot access 'a' before initialization
2 const a = 100;
```

# Scoping

- Scoping in JavaScript refers to the ***accessibility or visibility*** of variables in different parts of the code.
- Variables in JavaScript can be scoped ***globally, functionally, or block-level***, depending on how they are declared (var, let, or const).

## Scoping in var :

- ✓ var is **function-scoped**, meaning it is accessible throughout the function where it is declared, even before its declaration due to hoisting.
- ✓ However, it is **not block-scoped**, so it can "**leak**" outside block statements like if or for.

```
1  if (true) {  
2    |    var z = 50;  
3  }  
4  console.log(z); // 50 (Accessible outside the block)  
5
```

# Behavior of “var” keyword:

## Scoping in let :

let is block-scoped, meaning it is only accessible within the block, statement, or expression where it is defined.

```
1  if (true) {
2    let z = 50;
3    console.log(z); // 50 (Accessible inside the block)
4  }
5  console.log(z); // ❌ Error: z is not defined (Outside block)
```

## Scoping in const :

const is block-scoped, meaning it is only accessible within the block, statement, or expression where it is defined.

```
1  if (true) {
2    const z = 50;
3    console.log(z); // 50 (Accessible inside the block)
4  }
5  console.log(z); // ❌ Error: z is not defined (Outside block)
```



# Scoping

```
1 // ◆ Global Scope - Accessible anywhere
2 var globalVar = "I am a Global var";
3 let globallet = "I am a Global let";
4 const globalConst = "I am a Global const";
5
6 function demoScope() {
7   // ◆ Function Scope - Accessible only inside this function
8   var functionVar = "I am a Function var";
9   let functionLet = "I am a Function let";
10  const functionConst = "I am a Function const";
11
12  if (true) {
13    // ◆ Block Scope - Exists only inside this block {}
14    var blockVar = "I am a Block var"; // 🚩 NOT truly block-scoped (escapes block)
15    let blockLet = "I am a Block let"; // ✅ Block-scoped
16    const blockConst = "I am a Block const"; // ✅ Block-scoped
17
18    console.log(blockVar); // ✅ Accessible (but behaves differently)
19    console.log(blockLet); // ✅ Accessible (inside the block)
20    console.log(blockConst); // ✅ Accessible (inside the block)
21  }
22
23  console.log(blockVar); // ✅ Accessible (var escapes block scope)
24  // console.log(blockLet); ❌ ERROR - Not accessible outside the block
25  // console.log(blockConst); ❌ ERROR - Not accessible outside the block
26 }
27
28 demoScope();
29
30 console.log(globalVar); // ✅ Accessible
31 console.log(globallet); // ✅ Accessible
32 console.log(globalConst); // ✅ Accessible
33
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PLAYWRIGHT

▼ TERMINAL

```
● PS C:\Work Space\Practice\tests> node .\scoping.js
I am a Block var
I am a Block let
I am a Block const
I am a Block var
I am a Global var
I am a Global let
I am a Global const
```

# Scoping sample program to understand better

# Comparison Summary Table: var, let, and const

Feature	var	let	const
Re-declaration	✅ Allowed in the same scope	❌ Not allowed in the same scope	❌ Not allowed in the same scope
Re-assignment	✅ Allowed	✅ Allowed	❌ Not allowed
Scoping	Function-scoped or global	Block-scoped	Block-scoped
Block Awareness	❌ Does not respect block scope	✅ Respects block scope	✅ Respects block scope
Hoisting	✅ Hoisted and initialized to <code>undefined</code>	✅ Hoisted but uninitialized (TDZ applies)	✅ Hoisted but uninitialized (TDZ applies)
TDZ (Temporal Dead Zone)	❌ No TDZ	✅ Exists (cannot access before declaration)	✅ Exists (cannot access before declaration)

# Classroom on keyword var, let, const behavior :

## Assignment Details:

Declare a global variable and shadow it inside a function using both `var` and `let` to see how they behave differently when printed.

## Assignment Requirements:

1. Declare a const name as browserVersion (global)
2. Assign value as Chrome
3. Create a function by name getBrowserVersion
4. Create if condition inside function to check if browser is chrome, then
5. Declare a local variable (browserVersion) and print that variable inside function (outside block)
6. Call that function from the javascript

## Hints to Solve:

- Use 'var' first as block variable and then convert that as 'let'
- Confirm how it works

## Expected Outcome:

Upon completion, you should be able to:

- Understand the concepts of var, let and const and the hoisting principles