



# Full Stack Software Development

**Course:** JavaScript and  
Server-Side Communication

**Lecture On:** Basics of  
JavaScript

**Instructor:** Siddhesh  
Prabhugaonkar

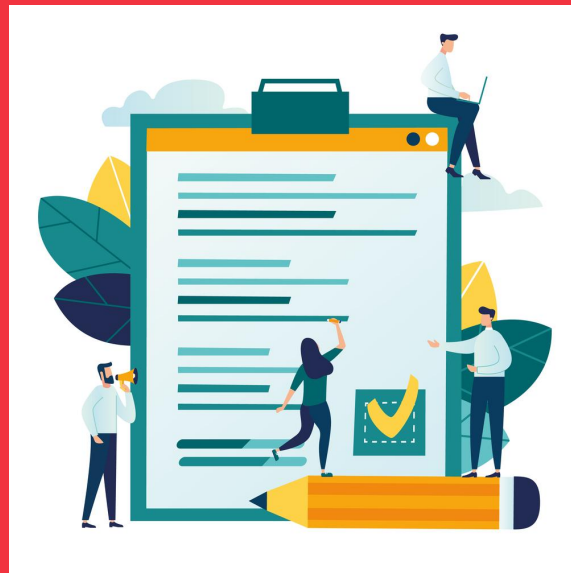


# In the previous course, we discussed...

- HTML
- CSS
- Bootstrap

# Today's Agenda

- Introduction to JavaScript (JS)
- How to add JS to HTML files
- Syntax of JS
- Operators in JS
- Looping and conditional statements in JS
- Error handling in JS

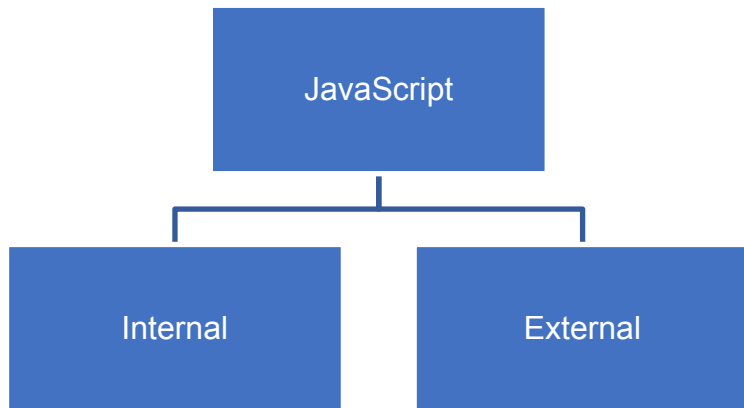


# Introduction to JavaScript



- JavaScript (JS) is the programming language of web development. It was initially used to 'make web pages alive'.
- JS programs are called scripts.
- Unlike Java, scripts do not require any compilation to run. JS can execute on a browser, a server and any device that has a 'JavaScript Engine'.
- JS does not provide low-level access to memory or CPU, which is why it is a 'safe' programming language.
- With the help of JS, you can perform the following:
  1. Adding or modifying HTML content on a page, including, but not restricted to, showing/hiding elements as well as changing the size, style and HTML attributes
  2. Reacting to user actions such as hovering, mouse clicks, key presses, etc.
  3. Sending requests to remote servers over the network and saving the data on the client side.

## Where to place scripts in HTML document



- JavaScript (JS) can be placed either internally in an HTML file using the `<script>` tag, where you can write all the scripts inside the `<script>` tag, or externally in a JavaScript file (that ends with a `.js` extension), which can be added as a source in the `<script>` tag.
- The `<script>` tag can be placed either in the `<head>` or in the `<body>` part of the HTML document.
- However, because JS is a [render blocker](#), it is preferable to call these scripts in the `<body>` tag after all the HTML elements are rendered.

## Where to place scripts in JavaScript

### Internal Stylesheet

index.html

```
<html>
  <head>
    <script>
      function onClick() { alert("Clicked"); }
    </script>
  </head>
  <body>
    <button onclick="onClick()">Click here.</button>
  </body>
</html>
```

### External Stylesheet

index.js

```
function onClick() {
  alert("Clicked");
}
```

```
<html>
  <head>
    <script src="index.js"></script>
  </head>
  <body>
    <button onclick="onClick()">Click
here.</button>
  </body>
</html>
```

index.html



# Poll 1 (15 Sec)

Which of the following is the syntactically INCORRECT method of including JavaScript into an HTML code?

1.

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // JavaScript Code
    </script>
  </head>
  <body>
  </body>
</html>
```

2.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <script>
    // JavaScript code
  </script>
  <body>
  </body>
</html>
```

3.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <script>
      // JavaScript Code
    </script>
  </body>
</html>
```

# Poll 1 (Answer)

Which of the following is the syntactically INCORRECT method of including JavaScript into an HTML code?

1.

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // JavaScript Code
    </script>
  </head>
  <body>
  </body>
</html>
```

2.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <script>
    // JavaScript code
  </script>
  <body>
  </body>
</html>
```

3.

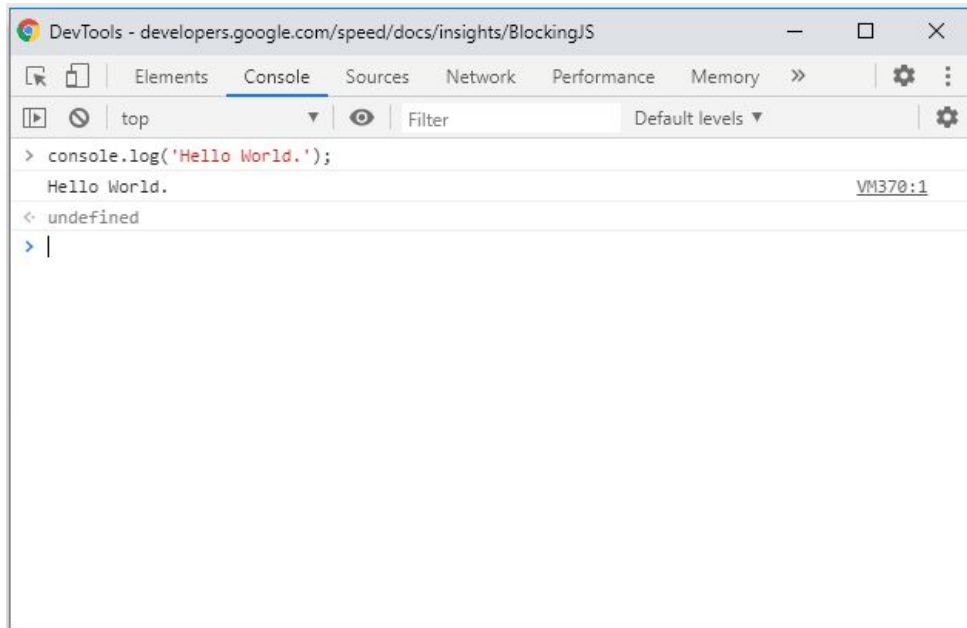
```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <script>
      // JavaScript Code
    </script>
  </body>
</html>
```

## JavaScript Output

JavaScript (JS) can display data in the following different ways. We shall start looking at the first method which is:

**Using console.log():** For debugging purposes, you can use **console.log()** to log the JS code in the browser's developer tools.

```
console.log('Hello World.');
```



## JavaScript Output

JavaScript (JS) can display data in the following four different ways:

1. **Writing into an HTML element:** Using **document.getElementById(id)**, you can fetch an element with an id, and then, using **innerHTML**, you can write into an HTML element.

```
document.getElementById('text1').innerHTML = "This is a paragraph" ;
```

2. **Writing into the HTML output:** Using **document.write()**, you can write into the document, i.e., the web page directly. Please note that when you use **document.write()**, the entire HTML will get deleted and replaced by the content inside the parentheses of **document.write()**.

```
document.write('Hello World.');
```

3. **Using window.alert():** Using **window.alert()**, you can display data in an alert box on the web page.

```
window.alert('Hello World.');
```

4. **Using console.log():** For debugging purposes, you can use **console.log()** to log the JS code in the browser's developer tools.

```
console.log('Hello World.');
```

# JavaScript Syntax

```
function clickMe() {  
    alert("I am clicked!");  
}
```

## JavaScript Terminology

- JavaScript (JS) programs are a list of programming instructions called **statements** that are executed by web browsers.
- JS statements consist of values, operators, expressions, keywords and comments. They are executed in a waterfall fashion, that is, one by one, in the order in which they are written.

## JavaScript Syntax

- Generally, a JavaScript (JS) code consists of statements with variables that store values or expressions.

```
var x, y; //Declaring variables
var num1 = 24; //Assigning values to variable
var num2 = 30; //Assigning values to variable
var total = num1 + num2; //Computing expressions and storing result in variable
```

- In a programming language, **variables** are used to store data values. In JS, variables are declared using the **var** keyword. An equal sign ('=') is used to assign value to the variables.
- JS **values** can be strings, numbers, arrays, objects, booleans or expressions. String values need to be written within single or double quotes, for example, 'John Doe' or "John Doe", and number values should not be written within quotes, for example, 2 or 200.45.

## JavaScript Syntax

- JS operators can be used to compute values; e.g.,

```
var sum = (24 + 200) * 50;
```

- JS **expressions** are a combination of values, variables and operators:

```
var mul = 10 * 10;  
var x = num1 * 10;  
var name = "John" + " " + "Doe";
```

- For single-line JS comments, you can use `//`, and for multiline comments, you can use `/* */`.



## JavaScript Terminology

- JavaScript (JS) programs are a list of programming instructions called **statements** that are executed by web browsers.
- JS statements consist of values, operators, expressions, keywords and comments. They are executed in a waterfall fashion, that is, one by one, in the order in which they are written.
- JS statements end with a semicolon (;). Semicolons essentially separate JS statements. It is not mandatory to add a semicolon, but it is highly recommended. Using semicolons, you can add multiple JS statements in one line.

Forexample,

```
var name = "John Doe";  
var age = 24;
```

## JavaScript Terminology

- JS ignores multiple spaces. However, it is recommended to add white spaces to your code to make it more readable. For example,

```
var name = "John";
```

..

- A best practice is to avoid having more than 80 characters in one line for better readability. The best place to break a JS code to a new line is immediately after the operator.

## JavaScript Variables: Identifiers and Data Types

- JavaScript (JS) variables are containers for storing data values. In earlier examples, you learnt that variables can store values and expressions.
- All JS variables must be identified with a unique name called **identifier**. The rules for writing an identifier name are as follows:
  1. Names can contain letters, digits, the dollar sign ('\$') and the underscore sign ('\_').
  2. Names should always begin with a letter.
  3. JavaScript is case-sensitive, i.e., *name* and *Name* are different variables.
  4. [Reserved JS words](#) cannot be used for an identifier.
  5. The best practice is to **not** keep a name, such as x, y, z, that does not indicate the meaning or significance of that variable. The identifier name should signify the meaning of the variable; for example, *name* and *age*.
- In JS, you can assign a value to the variable using the '=' operator.

## JavaScript Variables: Declaring Variables

Creating a variable in JavaScript (JS) is called **declaring a variable**. You can declare a JS variable using the **var** keyword.

```
var name;
```

When a variable is declared as shown in the example given above, its value is **undefined**. Use the assignment operator to assign a value to the variable. If the variable is already declared, then you need not use the keyword **var** again.

```
name = "John Doe";
```

You can also assign a value to a variable when declaring it.

```
var name = "John Doe";
```

## Poll 2 (15 Sec)

Suppose you declare a new variable called 'city' consisting of the value 'Berlin', which is as follows:

```
var city = "Berlin";
```

According to you, what will the output when you try to run the following command on the console?

```
console.log(city);
```

1. city
2. Berlin
3. ""
4. Error

## Poll 2 (Answer)

Suppose you declare a new variable called 'city' consisting of the value 'Berlin', which is as follows:

```
var city = "Berlin";
```

According to you, what will the output when you try to run the following command on the console?

```
console.log(city);
```

1. city
2. **Berlin**
3. " "
4. Error

## Poll 3 (15 Sec)

The variable 'city' that you declared in the previous question is as follows:

```
var city = "Berlin";
```

According to you, what will the output when you try to run the following command on the console?

```
console.log("city");
```

1. city
2. Berlin
3. ""
4. Error

# Poll 3 (Answer)

The variable city that we declared in the previous questions is as follows:

```
var city = "Berlin";
```

According to you, what will the output when you try to run the following command on the console?

```
console.log("city");
```

1. **city**
2. Berlin
3. ""
4. Error



## Primitive Data Types

- In JavaScript (JS), primitive data types refer to the data types that are used to store specific data of the same type, which means that they are not an object and do not have methods.
- The five primitive data types are as follows: String, Number, Boolean, undefined and null.
- All primitive values are immutable, i.e., they cannot be altered. You can conveniently assign a new value to a variable storing a primitive value, but you cannot change the value of a primitive data type.

123

Number

“A\*”

String

true  
false

Boolean

null  
undefined

# Strings

“The quick  
brown fox  
jumps over  
the lazy dog.”

## Primitive Data Types: Introduction to Strings

- [JavaScript \(JS\) strings](#) are used for storing and manipulating text.
- A JS string contains zero or more characters inside a string.
- You can use single or double quotes in a string/JS.

```
var firstName = "John";  
var lastName = 'Doe';
```

```
var x = "The nickname "Red Devils" is given to the players playing for Manchester United FC.";
```

In the example given above, JS engine will be confused when it encounters the second double quote before the word Red. Therefore, the string will be clipped to "The nickname". To avoid such a scenario, you can wrap the entire sentence in single quotes or use an escape character.

The backlash (\") or (\') escape character converts any special character into string characters.

\'	'
\"	"
\\	\

```
var x = "The nickname \"Red Devils\" is given to  
the players playing for Manchester United FC.";
```

## Primitive Data Types: String Methods

**length**: It returns the number of characters in a string (including spaces).

```
var str = "Hello World";  
console.log(str); //11
```

**indexOf()** and **lastIndexOf()**: **indexOf()** returns the position of the first occurrence of text in a string, whereas **lastIndexOf()** returns the last position of the occurrence of text in a string. You can even add a second parameter to signify the starting position for the search. If the text is not found in the string, then the value returned is -1.

```
var str = "I have to go to New York.";  
console.log(str.indexOf("to")); //7  
console.log(str.lastIndexOf("to")); //13  
console.log(str.indexOf("e", 4)); //5  
console.log(str.indexOf("Mumbai")); //-1
```

## Primitive Data Types: String Methods

### [slice\(start, end\):](#)

- It extracts part of the string and returns the extracted part in a new string without modifying the original string.
- The method takes two parameters, the *start position* and the *end position*;
- For instance, in the example given below, the start position is given as 7, and the end position is given as 12. So, the method will take the values from the positions 7 to 12.

```
var str = "I have to go to New York."; console.log(str.slice(7, 12)); //to go
```

If the second parameter is not given, then the method will slice out until the end of the string.

```
var str = "I have to go to New York."; console.log(str.slice(10)); //go to New York.
```

## Primitive Data Types: String Methods

**substring(start, end):** It is similar to **slice()**. The only difference is that **substring()** does not accept negative indices. If you omit the second parameter, then it will extract the rest of the string.

```
var str = "I have to go to New York."; console.log(str.substring(7, 12)); //to go
```

**substr(start, length):**

- **substr()** is also similar to **slice()** and **substring()**. The difference is that the second parameter is the length of the extracted part.
- It accepts a negative number in the first parameter, and that will start counting from behind.
- If the second parameter is not specified, then it will extract the remaining part of the string.

```
var str = "I have to go to New York."; console.log(str.substr(7, 5)); //to go
```

```
var str = "I have to go to New York."; console.log(str.substr(10)); //go to New York.
```

## Primitive Data Types: String Methods

### [replace\(text1, text2\):](#)

- The **replace()** method finds the first parameter in a string and replaces it with a second string.
- It does not change the string it is called on but returns a new string.

```
var str = "I have to go to New York.";
console.log(str.replace("New York", "Mumbai")); //I have to go to Mumbai.
```

- By default, the **replace()** method only replaces the first match.
- By default, the **replace()** method is case-sensitive. Writing NEW YORK will not return anything.
- To replace case-insensitiveness, use a regular expression with the `/i` flag.
- To replace all matches, use a regular expression with the `/g` flag.

```
var str = "I have to go to New York City in the New York state.";
console.log(str.replace("New York", "Mumbai")); //I have to go to Mumbai City in the New York state.
console.log(str.replace("NEW YORK", "Mumbai")); //
console.log(str.replace("/NEW YORK/i", "Mumbai")); //I have to go to Mumbai City in the New York state.
console.log(str.replace("/New York/g", "Mumbai")); //I have to go to Mumbai City in the Mumbai state.
```

## Primitive Data Types: String Methods

[toUpperCase\(\)](#) and [toLowerCase\(\)](#): The **toUpperCase()** method converts a string to the upper case, and the **toLowerCase()** method converts a string to the lower case.

```
var str = "Hello World";  
console.log(str.toUpperCase()); //HELLO WORLD  
console.log(str.toLowerCase()); //hello world
```

[concat\(\)](#): The **concat()** method joins two or more strings. It is the same as the '+' operator.

```
var str = "Hello"; console.log(str.concat(" ", "World")); //Hello World
```

[trim\(\)](#): The **trim()** method removes white spaces from the string.

```
var str = "    Hello.    "; console.log(str.trim()); //Hello
```



## Primitive Data Types: String Methods

[charAt\(position\)](#): The **charAt()** method returns the character at a specified index (position) in a string. The first position is always 0.

```
var str = "Hello World"; console.log(str.charAt(2)); //l
```

**[position]**: Using [], you can access the character at that position.

```
var str = "Hello World"; console.log(str[0]); //H
```

# Numbers



## Primitive Data Types: Introduction to Number

- JavaScript (JS) has the datatype named [Number](#), which encompasses all kinds of numbers. It can be written with or without decimals.
- Numbers are written without quotes.
- Numbers that are written with quotes are **not** numbers but strings.

```
var num1 = 31;  
var num2 = 31.423;
```

```
var num1 = 5, num2 = 4;  
var sum = num1 + num2; //9
```

```
var num1 = "5", num2 = "4";  
var sum = num1 + num2; //54
```

```
var num1 = "5", num2 = "4";  
var sum = num1 * num2; //20
```

```
var num1 = "5", num2 = "4";  
var sum = "Result: " + x +  
y; //Result54
```

```
var num1 = 5, num2 = "4";  
var sum = 1 + x + y; //64
```

## Primitive Data Types: NaN and Infinity

- [NaN](#) is a JavaScript-reserved keyword for notifying that a number is not a legal number. If you try to perform arithmetic operations with a non-numeric string, the result will be NaN.

```
var x = 4;  
var y = "Apple";  
var z = x / y; //NaN
```

- [Infinity or -Infinity](#) is the value returned when the calculation results in a number more than the largest possible number. Dividing a positive number by zero results in infinity. Dividing a negative number by 0 results in -Infinity.

```
var x = 4;  
var y = 0;  
var z = x / y; //Infinity
```

## Primitive Data Types: Number Methods

- Before you take a look at number methods, **typeof** is an operator that helps in identifying the data type of a value.

```
var x = typeof 4; //number
```

```
var y = typeof "Apple"; //string
```

---

**toString()**: The toString() method returns a number as a string. It can change any number, even if the method is inside a variable or is an expression.

```
var x = 200;  
console.log(typeof x.toString(), x.toString()); //string 200  
console.log(typeof (123).toString(), (123).toString()); //string 123  
console.log(typeof (123+100).toString(), (123+100).toString()); //string 223
```

---

**toFixed()**: The toFixed() method returns the number as a **string** with a specified number of decimals.

```
var x = 10.639;  
console.log(x.toFixed(0)); //11  
console.log(x.toFixed(1)); //10.6  
console.log(x.toFixed(2)); //10.64
```

## Primitive Data Types: Number Methods

**Number()**: The Number() method returns a number. It converts numeric strings into numbers, dates into milliseconds since 1 January 1970, and true/false as 1/0 respectively.

```
Number(true); //1
Number(false); //0
Number("10"); //10
Number(" 10 "); //10
```

```
Number("10.33"); //10.33
Number("10,33"); //NaN
Number("John"); //NaN
Number(new Date("06-10-2020")); //1591727400000
```

**parseInt()**: The parseInt() method parses a string and returns a whole number. The parsed number is returned till the point where the first non-whitespace character cannot be converted to a number.

```
parseInt("10"); //10
parseInt("10.33"); //10
parseInt("10 20 30"); //10
parseInt("10 years"); //10
```

**parseFloat()**: The parseFloat() method parses a string and returns a floating-point number.

```
parseFloat("10"); //10
parseFloat("10.33"); //10.33
parseFloat("10 years"); //10
```

## Poll 4 (15 Sec)

What will be the output of the following code on the console?

```
var x = parseFloat("10.3")  
console.log(x);
```

1. 10
2. 10.3
3. 10.0
4. NaN

# Poll 4 (Answer)

What will be the output of the following code on the console?

```
var x = parseFloat("10.3")  
console.log(x);
```

1. 10
2. **10.3**
3. 10.0
4. NaN



# Boolean

```
var firstNum = 1;  
var secondNum = 2;  
var newNum = 1;  
  
var booleanVariable1 = true;  
var booleanVariable2 = false;
```

# undefined and null

```
var x;  
console.log(x) //undefined  
  
var y = null;  
console.log(y) //null
```

## Primitive Data Types: undefined and null

- In JavaScript (JS), a variable without any value is of **undefined** type.
- You can deliberately assign the value undefined to any variable in JS.
- However, an empty value is not undefined. `var firstName = "";` is not undefined, and it has some value, which is empty string.
- [null is a type in itself although typeof null is object.](#) In JS, null is nothing. It is supposed to be something that does not exist.

Observe [this](#) image for better understanding

## Poll 5 (15 Sec)

Which of the following is true regarding **undefined** in JavaScript?

1. The variable that is used in the code is not declared.
2. The variable is declared but is not given any value.
3. The mentioned property does not exist in the code.
4. The variable is assigned a null value.

## Poll 5 (Answer)

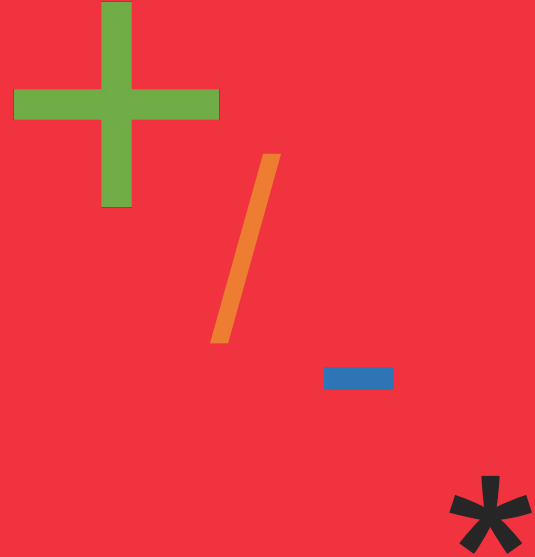
Which of the following is true regarding **undefined** in JavaScript?

1. The variable that is used in the code is not declared.
2. **The variable is declared but is not given any value.**

'Undefined' means that the variable has been declared, but you have not assigned it a value yet. In such a case, when you try to print the value of this variable on to the console, it says 'undefined', i.e., you have not defined any value of this variable yet.

1. The mentioned property does not exist in the code.
2. The variable is assigned a null value.

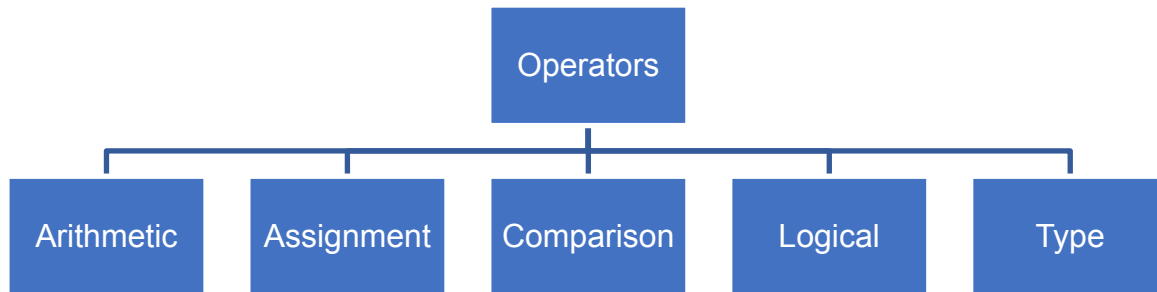
# Operators



## Different Types of Operators

There are different types of operators: you can find the entire list [here](#).

We would be covering five types of operators: arithmetic (to perform arithmetic operations on numbers), assignment (to assign values to JavaScript variables), comparison (to compare values), logical (to determine the logical relationship between variables/values) and type (to identify the datatype of variables).



## Arithmetic Operators

[Arithmetic operators](#) help in performing arithmetic operations on JavaScript (JS) numbers. Arithmetic operations can be performed between values and variables, and they can also include expressions.

```
var x = 5, y = 2;
console.log(x + y); //7
console.log(x - y); //3
console.log(x * y); //10
console.log(x / y); //2.5
console.log(x % y); //1    /* % is modulus operator which calculates the remainder of division
x++;
console.log(x); //6
x--;
console.log(x); //5
console.log(x ** y); //25 /* ** is exponential operator */
console.log(x ** y/x); //5 /* ** is has more precedence as compared to / */
```

**Operator precedence:**  
parentheses > division > multiplication > addition > subtraction



## Poll 6 (15 Sec)

What will be the result of the following operation in JavaScript?

```
console.log(2 ** 4 / 4);
```

1. 2
2. 4
3. 8
4. 16

# Poll 6 (Answer)

What will be the result of the following operation in JavaScript?

```
console.log(2 ** 4 / 4);
```

1. 2

2. 4

3. 8

4. 16

## Assignment Operators

Compound assignment operators are shorthand operators to assign values to JavaScript variables after doing some operation.

```
var x = 5;    //Assigns value 5 to x
console.log(x += 5); //10    /* This is similar to x = x + 5 */
console.log(x -= 2); //8    /* This is similar to x = x - 2. Since x's latest value is 10, the answer will be 8. */
console.log(x *= 2); //16    /* This is similar to x = x * 2 */
console.log(x /= 2); //8    /* This is similar to x = x / 2 */
console.log(x %= 2); //0    /* This is similar to x = x % 2 */

// Here, + works for concatenation.
var greeting = "Hello";
console.log(greeting += " World"); //Hello World
```

## Comparison Operators

[Comparison operators](#) are used in logical statements to determine equality or difference between variables or values.

The comparison operators are as follows:

1. **> and >=**: These operators are used to compare two numbers and determine which number is greater. >= also includes the number to the right of the operator.
2. **< and <=**: These operators are used to compare two numbers and determine which number is smaller. <= also includes the number to the right of the operator.
3. **==**: This operator is used to check whether two numbers are equal. Therefore, if you compare a number with a numeric string where both have the same value, then it will return true.
4. **===**: This operator is a stricter check for equality. It also checks for types of the variables. Therefore, if you compare a number with a numeric string, then it will return false.
5. **!=**: This operator is used to check whether two numbers are not equal.
6. **!==**: This operator is a stricter check for non-equality. Like ===, it also checks for types of the variables.

```
var x = 2;    //Assigns value 2 to x
console.log(x == 2); //true
console.log(x == 8); //false
console.log(x == "2"); //true
console.log(x === 2); //true
console.log(x === "2"); //false

console.log(x != 5); //true
console.log(x != 2); //false
console.log(x !== "2"); //true

console.log(x > 0); //true
console.log(x > 2); //false
console.log(x >= 2); //true
console.log(x < 5); //true
console.log(x < 2); //false
console.log(x <= 2); //true
```

# Poll 7 (15 Sec)

What will be the output of the following program?

```
0 == false
```

1. True
2. False
3. Invalid operation
4. None of the above

# Poll 7 (Answer)

What will be the output of the following program?

```
0 == false
```

1. **True**
2. ☐ False
3. Invalid operation
4. None of the above

# Poll 8 (15 Sec)

What will be the output of the following program?

```
null == undefined
```

1. True
2. ☐ False

# Poll 8 (Answer)

What will be the output of the following program?

```
null == undefined
```

1. **True**

Read point 2 [here](#).

1. **False**



# Poll 9 (15 Sec)

What will be the output of the following program?

```
null === undefined
```

1. True
2. ☐ False

# Poll 9 (Answer)

What will be the output of the following program?

```
null === undefined
```

1. True
2. **False**

When two quantities are compared using the "===" operator, both the value and the data type of the two quantities are checked to see whether they are equal. Thus, here, both the value and the data type of null and undefined will be checked to see whether they are equal. As their data types do not match, the result will be false.

## Logical Operators

[Logical operators](#) are used in logical statements to determine the logic between two or more expressions, values or variables. The logical operators are as follows:

1. **&&:** This operator returns true only when **all** the expressions are true.
2. **||:** This operator returns true only if **either** of the expressions is true.
3. **!:** This operator returns true for false statements and false for true statements.

```
var x = 2, y = 6;
console.log(x > 0 && y < 10); //true
console.log(x > 5 && y < 10); //false

console.log(x > 0 || y < 10); //true
console.log(x > 5 || y < 10); //true
console.log(x > 5 && y < 4); //false

console.log(!(x == 5)); //true
console.log(!(x >= 5)); //true
```

## Type Operators

Type operator are used to identify the data type of a variable.

```
var x = 2, y = "Hello", z = true;
var a;
var b = null, c = 2/0, d = 2/"Hello";
var e = { name: "Mercedes", model: "C200", color: "white", weight: 500 };
var f = [ "Mercedes", "C200", "white", 500 ];

console.log(typeof x); //number
console.log(typeof y); //string
console.log(typeof z); //boolean
console.log(typeof a); //undefined
console.log(typeof b); //object
console.log(typeof c); //number
console.log(typeof d); //number /* NaN is a type number */
console.log(typeof e); //object
console.log(typeof f); //object
```

## Type Conversion

In JavaScript (JS), you can convert variables from one type to another using JavaScript functions. Also, sometimes JS itself converts the datatype of the variables.

**Converting numbers to strings:** You can use the global method **String()** or the **toString()** method. Even when you use number methods, such as the **toFixed()** method or the **toPrecision()** method, the value gets converted to a string.

```
var x = 2.56;
console.log(typeof String(x)); //string
console.log(typeof x.toString()); //string
console.log(typeof x.toFixed(1)); //string
```

**Converting strings and booleans to numbers:** Numeric strings can be converted to numbers. On the other hand, the conversion of non-numeric strings to numbers will result in **NaN**. You can use the global method **Number()** or use a unary operator, such as **+**, before a string to convert it to a number. Also, using methods, such as **parseInt()** and **parseFloat()** methods, will return numbers. For boolean, the **Number()** method can convert a boolean to a number.

```
var x = "2.56", y = "John", z = true;
console.log(typeof Number(x)); //number
console.log(typeof +x); //number
console.log(typeof parseInt(x)); //number
console.log(Number(y)); //NaN
console.log(typeof Number(y)); //number
console.log(typeof Number(z)); //number
```

## Automatic Type Conversion

- When JavaScript tries to operate on a “wrong” data type, it automatically converts it to a “right” data type.
- When you try to output an object or a variable, the **toString()** function is automatically called, and the object/variable is converted to a string.

```
console.log(10 + null); //10  /* Because null is automatically converted to 0 */
console.log("10" + null); //10null. /* Because null is automatically converted to string "null" */
console.log("10" + 5); //105  /* Because 10 is a string and the operation is +, it is considered as concatenation
and 5 is automatically converted to string "5" */
console.log("10" - 5); //5    /* Because 10 is a string but the operation is -, 10 is automatically converted to
number 10 */
console.log("10" * "5"); //50  /* Because both 10 and 5 are strings and the operation is not +, the two strings are
converted to numbers and multiplied. */
```

# Type Coercion

When you add a string and numbers, if the string is declared first, then the remaining values (even if they are numbers) will be considered to be strings.

```
var x = "5" + 2 + 3; //523  
var y = "HTML" + 5; //HTML5
```

However, if numbers are declared first, then the remaining values are considered to be numbers until they reach a string

```
var x = 5 + 2 + "3"; //73  
var y = 2 + 3 + "HTML"; //5HTML
```

You can read more about type coercion [here](#) and [here](#).



# Poll 10 (15 Sec)

What will be the output of the following program?

```
var x = 5 + true;  
console.log(x);
```

1. true
2. false
3. 5
4. 6

# Poll 10 (Answer)

What will be the output of the following program?

```
var x = 5 + true;  
console.log(x);
```

1. true

2. false

3. 5

4. 6

# Doubts Clearance (5 min)

# JavaScript Conditions

if ()



else ()



## Introduction to Conditional Statements

- Conditional statements help JavaScript perform different actions based on certain given conditions.
- There are two types of conditional statements in JS: **if...else** and **switch()**.

if ()



else ()



switch ()

case 1:



case 2:



## If...else

- The first variant is the **if** conditional statement. It checks whether a specified condition in a parameter is true or false. If it is true, then it executes the code corresponding to the condition, and if it is false, it does not execute the condition inside the block.
- Another variant is the **if...else** conditional statement. If the condition in the if statement is true, then the code will execute that condition. However, if it is false, then it will execute the else block.

```
var x = 2;

if(x % 2 == 0) {
    console.log("The number is even");
}

//The number is even
```

```
var x = 3;

if(x % 2 == 0) {
    console.log("The number is even");
} else {
    console.log("The number is odd");
}

//The number is odd
```

```
var x = "Hello";

if(x % 2 == 0) {
    console.log("The number is even");
} else if (x % 2 == 1){
    console.log("The number is odd");
} else {
    console.log("x is not a number");
}

//x is not a number
```

## Switch

The **switch** statement can be used to perform different actions based on different conditions. In the switch parentheses, an expression needs to be added. The answer to that expression evaluation will be one of the **cases** inside the switch statement. If none of the cases match, then the code should go to **default**. The switch statement is like a waterfall: If the first case matches the value, then it executes the code written inside the first case but it will go to all the cases and execute statements in them. To avoid this, every case should end with a **break**. The **break** keyword stops the execution at that point and comes out of the switch.

```
var x = 3;
switch(x - 2) {
  case 0:
    console.log("The answer is zero");
    break;
  case 1:
    console.log("The answer is one");
    break;
  case 2:
    console.log("The answer is two");
    break;
  default:
    console.log("x is not a number");
}
// The answer is one
```

## Poll 11 (15 Sec)

What will be the output of the following program on the console?

1. 'Play Beethoven'
2. 'Play Bob Dylan'
3. Invalid operation
4. No output

```
var expr = 'electric guitar';
switch (expr){
  case 'piano':
    console.log('Play Beethoven');
    break;
  case 'acoustic guitar':
    console.log('Play Bob Dylan');
    break;
}
```



# Poll 11 (Answer)

What will be the output of the following program on the console?

1. 'Play Beethoven'
2. 'Play Bob Dylan'
3. Invalid operation
4. **No output**

```
var expr = 'electric guitar';  
switch (expr){  
    case 'piano':  
        console.log('Play Beethoven');  
        break;  
    case 'acoustic guitar':  
        console.log('Play Bob Dylan');  
        break;  
}
```

The switch statement evaluates an expression by matching its value with a case clause and executes statements associated with that case as well as statements in cases that follow the matching case (unless that case is followed by the "break" keyword). In the given program, as no case matches the switch statement, nothing will be printed on the console.

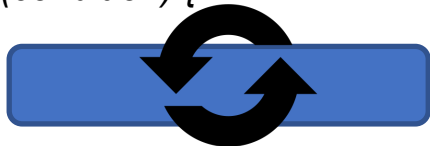
# JavaScript Looping



## Introduction to Loops

- Loops can execute a block of code several times.
- In JavaScript, there are three different methods for looping: **for()**, **while()**, and **do...while()**.

for (*condition*) {



}

while (*condition*) {



}

do {



} while (*condition*)

## The for loop

Let's assume that you have an array and you want to print every element in the array on a different line. You would have to run the same printing statement again and again, with different indices of the array.

```
var students = [ "Rajesh", "Sheldon", "Leonard", "Howard"];

var text += students[0] + "\n";
text += students[1] + "\n";
text += students[2] + "\n";
text += students[3] + "\n";
console.log(text);
```

```
Rajesh
Sheldon
Leonard
Howard
```

Output

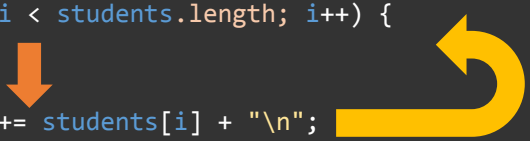
In this case, it was a small array of four elements. However, if you have an array of 100 elements, then the process becomes tedious. This is where for loop comes into the picture.

The syntax of a for loop has three statements: **statement1**, used for initialising variables (it is executed before the execution of the code block); **statement2**, which is for condition (it defines the condition for executing the code block); and **statement3**, which is for incrementing/decrementing the value initialised in statement1 (it is executed every time the code block is executed)

```
for(initialization; condition; updation) {
    //code block to execute
}
```

## The for loop

```
var students = [ "Rajesh", "Sheldon", "Leonard", "Howard"];  
  
for(var i = 0; i < students.length; i++) {  
    var text += students[i] + "\n";  
}
```



- As shown in the given code, the first step is to initialisation of the variable `i` with a value 0.
- Next, it will check the condition in statement 2. If the value is true, then it will execute the code block. Since  $0 < \text{length of student array } (=4)$ , the condition is true.
- Then, the statement in the code block is executed, where `students[i]`, which will be `students[0]` as `i = 0`, will be added to `text`. Hence, "Rajesh" is appended to `text`.
- Then, the increment will happen as part of updation statement inside the for loop; so, the value of `i` will become 1.
- Next, the condition will be checked again, and if it is true, then it will execute the code block. If it is false, which means that the entire array has been traversed, then it will stop the execution and come out of the for loop.

## The while loop

The **while** loop loops through a block of code as long as the specified condition holds true.

```
while(condition) {  
    //code block to execute  
}
```

In simple words, the code block will keep executing as long as the condition mentioned in the parentheses does not turn false. As shown in the example given below, the code will keep adding value to the text as long as the value of *i* remains below 10. An important point to note here is that if the value of *i* is never incremented, then the loop will keep executing an infinite number of times, which will crash your browser. So, it is important to ensure that there is always an increment/decrement of value.

```
var i = 0, text="";  
while(i < 10) {  
    text += "The number is " + i + "\n";  
    i++;  
}  
console.log(text);
```

## The do...while loop

The **do...while** loop is similar to the while loop. In the while loop, if the condition is not met, then the code block will not be executed at all. However, in the do...while loop, the code block will be executed once, and then the condition will be checked. If the condition is true, then the code block will be executed repeatedly until the condition becomes false.

```
do {  
    //code block to execute  
}  
while(condition);
```

In the first example given below, the condition is false. However, the code will be executed once, and outputting `text` will give "The number is 0". In the second example, as the condition is true, the code block will be executed until the condition becomes false. As with the while loop, if the variable is not incremented/decremented, then the browser will crash because of infinite loop.

```
var i = 0, text="";  
do {  
    text += "The number is " + i;  
    i++;  
}  
while(i > 10);  
console.log(text);
```

```
var i = 0, text="";  
do {  
    text += "The number is " + i;  
    i++;  
}  
while(i < 10);  
console.log(text);
```

# Error Handling





## Introduction to Error Handling

- Errors can occur during the execution of JavaScript code.
- It could be a
  - **SyntaxError** (for instance, when the starting quote is double quotes but the ending quote is a single quote, such as *"Hello"*),
  - **TypeError** (for instance, when you try to apply a string method to a number.
  - **ReferenceError** (for instance, when you try to use a variable in an expression without it being declared).
  - **RangeError** (for instance, when you try to use a number that is outside the range of legal values).
  - **URIError** (for instance, when unknown characters are used in the URI function).
- If these errors are not handled, then the application might crash, and users will see a blank page or an ugly error page on their screens.
- On the other hand, if these errors are caught during execution, then they could be handled and redirected as an alert box, or a custom error page, without the application crashing.
- These errors can be handled using **try**, **catch**, **throw** and **finally**.

## try, catch, throw and finally

The **try** statement allows you to test a code block for errors. The **catch** statement enables you to handle the errors caught in the try statement. The **throw** statement allows you to create custom errors. The **finally** statement executes the code after try and catch statements regardless of whether an error is caught or not.

```
try {  
    //Block of code to catch errors  
} catch(error) {  
    //Block of code to handle caught errors  
}
```

```
try {  
    alert("Hello");  
} catch(error) {  
    console.log(error);  
} //Uncaught SyntaxError: Invalid or unexpected token
```

```
try {  
    var x = 5;  
    if(x%2 !== 0)  
        throw "Not an even number";  
} catch(error) {  
    console.log(error);  
} finally {  
    console.log("The code is finally over");  
}  
// Not an even number  
// The code is finally over
```

# Hands-On Exercise (2 min)

What will the console output for the below JS statements.

```
var x = 5 + 2 + "3";  
var x = "5" + "2" + 3;  
var y = 2 + 3 + "HTML";  
var y = "HTML" + 5;
```

Solution:

```
var x = 5 + 2 + "3"; //73  
var x = "5" + "2" + 3; //523  
var y = 2 + 3 + "HTML"; //5HTML  
var y = "HTML" + 5; //HTML5
```

# Key Takeaways

- JavaScript (JS) is a programming language that is used for web development. JS programs are called scripts. They can be introduced internally or externally via `<script>` tags.
- JS has two data types: Primitive data types, such as string, number, boolean and undefined; and user-defined data types, such as objects and arrays.
- JS variables are used for storing data values.
- JS has different types of operators, such as: Arithmetic operators (for performing arithmetic operations), assignment operators (for assigning values to variables), comparison operators (for comparing two or more variables), logical operators (for determining the logic between variables).
- Conditional statements execute code based on conditions, looping statements execute a code block multiple times, and error handling code handles errors.

# Tasks to complete after today's session

MCQs
Coding Questions

## In the next class, we will discuss...

- Arrays in JavaScript
- Objects in JavaScript
- Functions in JavaScript



Thank you!