

1.1 Introduction to Javascript

Table of Content

1. What is Javascript
2. Variables & Data Types
3. Javascript Type Conversion
4. Javascript Arithmetic Operators
5. Conditional statements in Javascript

1. What is Javascript

JavaScript (JS) is a **high-level, interpreted, dynamic programming language** primarily used to create interactive and dynamic web content. It is one of the core technologies of the web, alongside HTML and CSS.

How Does JavaScript Work?

1. **Execution Environment:** JavaScript runs in the browser's JavaScript engine (like Google's V8 in Chrome, SpiderMonkey in Firefox, or JavaScriptCore in Safari). It can also run server-side using environments like Node.js.
2. **Interpreted Nature:** JavaScript code is directly executed by the engine, line by line. However, modern engines use **Just-In-Time (JIT) compilation** for better performance.
3. **Single-threaded Model:** JavaScript runs in a single thread but uses asynchronous mechanisms (like callbacks, promises, and async/await) to handle tasks like API calls or timers.

Compiler or Interpreter?

JavaScript is **both interpreted and compiled**, depending on the context:

- **Traditionally interpreted:** JavaScript engines are used to interpret the code line by line directly.

- **Modern engines use JIT compilation:** Engines like V8, Chakra, and SpiderMonkey compile JavaScript to optimize machine code at runtime (Just-In-Time) for faster execution.

Aspect	Java	JavaScript
Type	Compiled, statically typed.	Interpreted (or JIT compiled), dynamically typed.
Usage	General-purpose language for backend, desktop, mobile, etc.	Primarily used for web development to create dynamic and interactive web pages.
Execution	Requires JVM for execution.	Runs in web browsers (e.g., V8, SpiderMonkey) or server-side with Node.js.
Paradigm	Strongly object-oriented, with support for functional programming.	Multi-paradigm: procedural, functional, and prototype-based object-oriented programming.
Threading	Supports multi-threading with synchronization.	Single-threaded, event-driven, but uses async features (e.g., Promises, async/await).

Learn more: <https://dev.to/suprabhasupi/how-javascript-works-4ked>

▼ Where to put the Javascript code?

In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.

```
<script>
document.getElementById("demo").innerHTML = "My First Java";
</script>
```

You can place any number of scripts in an HTML document.

Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both.

We can also create separate External Javascript.

```
<script src="myScript1.js"></script>
```

▼ Keywords

In JavaScript, you cannot use these reserved words as variables, labels, or function names:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

▼ Javascript Statements

A **computer program** is a list of "instructions" to be "executed" by a computer.

In a programming language, these programming instructions are called **statements**.

A **JavaScript program** is a list of programming **statements**.

```
let a, b, c; // Declare 3 variables
a = 5;       // Assign the value 5 to a
b = 6;       // Assign the value 6 to b
c = a + b;   // Assign the sum of a and b to c
```

2. Variables & Data Types

▼ Variables

Variables are containers for storing data (storing data values). We can declare a variable by using these keywords.

- Using `var` for declaring function-scoped variables (old)
- Using `let` for declaring block-scoped variables (new)
- Using `const` for declaring constant variables

Note: It is recommended we use `let` instead of `var`. However, there are a few browsers that do not support `let`.

Rules for naming variables:

1. Variable names must start with either a letter, an underscore `_`, or the dollar sign `$`. For example,

```
//valid
let a = 'hello';
let _a = 'hello';
let $a = 'hello';
```

2. Variable names cannot start with numbers. For example,

```
//invalid
Let 1a = 'hello'; // this gives an error
```

3. JavaScript is case-sensitive. For example,

```
let y = "hi";
let Y = 5;

console.log(y); // hi
console.log(Y); // 5
```

4. Keywords cannot be used as variable names. For example,

```
//invalid
let new = 5; // Error! new is a keyword.
```

▼ Data Types

A data type, in programming, is a classification that specifies which type of value a variable has and what type of mathematical, relational or logical operations can be applied to it without causing an error.

A string, for example, is a data type that is used to classify text and an integer is a data type used to classify whole numbers.

Some Common Data Types in Javascript are:

- `String` for "Hello", 'hi' etc.
- `Number` for 45, -12 etc.
- `Boolean` for true and false
- `undefined` for un-initialized variables
- `Object` key-value pairs of collection of data

Aspect	Java	JavaScript
Variable Declaration	Use keywords like <code>int</code> , <code>String</code> , <code>double</code> , etc., to define a variable's type.	Uses <code>var</code> , <code>let</code> , or <code>const</code> , and types are inferred dynamically.
Typing	Statically typed (type must be defined at compile-time).	Dynamically typed (type is determined at runtime).
Primitives	<code>int</code> , <code>float</code> , <code>char</code> , <code>boolean</code> , etc., and corresponding wrapper classes (e.g., <code>Integer</code>).	Primitives: <code>Number</code> , <code>String</code> , <code>Boolean</code> , <code>Symbol</code> , <code>BigInt</code> , <code>undefined</code> , <code>null</code> .
Objects	Everything apart from primitives is a class or object (e.g., <code>ArrayList</code> , <code>HashMap</code>).	Objects are key-value pairs and include arrays, functions, and custom objects.

3. Javascript Type Conversion

We use these functions to convert types:

- `Number()`
- `String()`
- `Boolean()`

Note:

1. JavaScript considers 0 as false and all non-zero numbers as true. And, if true is converted to a number, the result is always 1.
2. String() takes `null` and `undefined` and converts them to string.
3. In JavaScript, `undefined`, `null`, `0`, `NaN`, `''` converts to false. All other values give `true`.

Use this table for reference:

Value	String Conversion	Number Conversion	Boolean Conversion
1	"1"	1	<code>true</code>
0	"0"	0	<code>false</code>
"1"	"1"	1	<code>true</code>
"0"	<code>"0"</code>	0	<code>true</code>
"ten"	<code>"ten"</code>	<code>NaN</code>	<code>true</code>
<code>true</code>	"true"	1	<code>true</code>
<code>false</code>	"false"	0	<code>false</code>
<code>null</code>	"null"	0	<code>false</code>
<code>undefined</code>	"undefined"	<code>NaN</code>	<code>false</code>
"	""	0	<code>false</code>
' '	" "	0	<code>true</code>

4. Javascript Arithmetic Operators

As with algebra, you can do arithmetic with JavaScript variables, using operators like = and +

```
const number = 3 + 5; // 8
```

We have Arithmetic Operators : +, -, /, *, ++, — and **

Aspect	Java	JavaScript
Operators	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>++</code> , <code>--</code> .	Same operators as Java.
Behavior	Arithmetic operates on strongly typed variables.	Works on numbers but performs type coercion.
Special Cases	Division by zero throws an exception.	Division by zero results in <code>Infinity</code> or <code>-Infinity</code> .

5. Javascript Comparison Operators

Operator	Description	Example
<code>==</code>	Equal to: <code>true</code> if the operands are equal	<code>5==5; //true</code>
<code>!=</code>	Not equal to: <code>true</code> if the operands are not equal	<code>5!=5; //false</code>
<code>===</code>	Strict equal to: <code>true</code> if the operands are equal and of the same type	<code>5==='5'; //false</code>
<code>!==</code>	Strict not equal to: <code>true</code> if the operands are equal but of different type or not equal at all	<code>5!=='5'; //true</code>
<code>></code>	Greater than: <code>true</code> if the left operand is greater than the right operand	<code>3>2; //true</code>
<code>>=</code>	Greater than or equal to: <code>true</code> if the left operand is greater than or equal to the right operand	<code>3>=3; //true</code>
<code><</code>	Less than: <code>true</code> if the left operand is less than the right operand	<code>3<2; //false</code>
<code><=</code>	Less than or equal to: <code>true</code> if the left operand is less than or equal to the right operand	<code>2<=2; //true</code>

▼ ternary Operator

A ternary operator evaluates a condition and executes a block of code based on the condition.

Its syntax is:

```
condition ? expression1 : expression2
let result = (marks >= 40) ? 'pass' : 'fail';
```

The ternary operator evaluates the test condition.

- If the condition is `true`, **expression1** is executed.
- If the condition is `false`, **expression2** is executed.

The ternary operator takes **three** operands, hence, the name ternary operator. It is also known as a conditional operator.

▼ If-else, else-if

In computer programming, there may arise situations where you have to run a block of code among more than one alternatives. For example, assigning grades **A**, **B** or **C** based on marks obtained by a student.

In such situations, you can use the JavaScript `if...else` statement to create a program that can make decisions.

```
if (condition) {
    // block of code if condition is true
} else {
    // block of code if condition is false
}
```

You can also write multiple `else if` in between the `if` and the `else` blocks.

▼ logical Operators

Operator	Description	Example
&&	Logical AND: <code>true</code> if both the operands/boolean values are true, else evaluates to <code>false</code>	<code>true && false; // false</code>
	Logical OR: <code>true</code> if either of the operands/boolean values is <code>true</code> , evaluates to <code>false</code> if both are <code>false</code>	<code>true false; // true</code>
!	Logical NOT: <code>true</code> if the operand is <code>false</code> and vice-versa.	<code>!true; // false</code>

▼ Switch Statements

The JavaScript `switch` statement is used in decision making.

The `switch` statement evaluates an expression and executes the corresponding body that matches the expression's result.

```
// program using switch statement
let a = 2;

switch (a) {
  case 1:
    a = 'one';
    break;
  case 2:
    a = 'two';
    break;
  default:
    a = 'not found';
    break;
}
console.log(`The value is ${a}`);
```

Aspect	Java	JavaScript
Statements	<code>if</code> , <code>else if</code> , <code>else</code> , <code>switch</code> .	Same structure: <code>if</code> , <code>else if</code> , <code>else</code> , <code>switch</code> .
Condition Syntax	Conditions must be explicitly boolean (e.g., <code>if (x > 0)</code>).	Non-boolean values are coerced into boolean (<code>if (")</code> evaluates as false).
Ternary Operator	<code>condition ? expr1 : expr2</code> .	Same operator.

Assignments

1. Build a Calculator Application (without the UI) using Arithmetic operators
2. Build an Average Marks Generator. using Arithmetic operators
3. Build a BMI calculator using Arithmetic operators
4. Build a program to grade students based on marks using switch-case