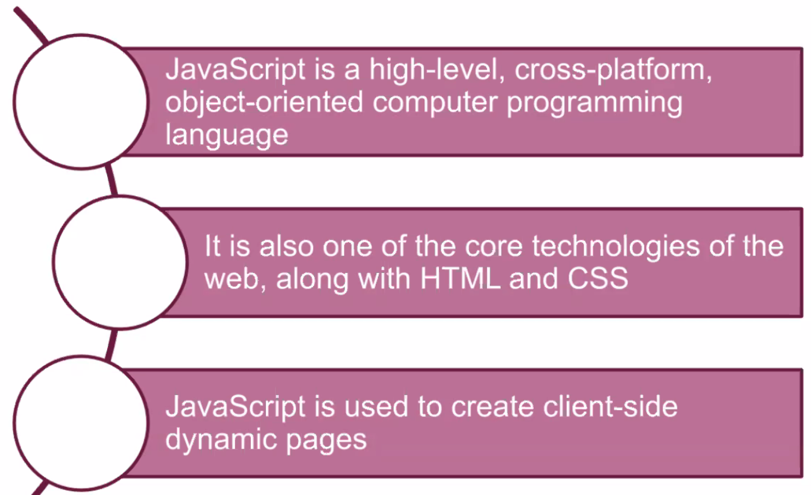


# What is Javascript?



**➤** **JavaScript** is a high-level, cross-platform, object-oriented computer programming language.

**➤** It is also one of the core technologies of the web, along with **HTML** and **CSS**.

**➤** **JavaScript** is the most commonly used to create client-side dynamic pages.

**➤** Today, **JavaScript** can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

**⚜️ Why is it called *Java*Script?:**

When JavaScript was created, it initially had another name: **“LiveScript”**. But Java was very popular at that time, so it was decided that positioning a new language as a “younger brother” of Java would help.  
But as it evolved, JavaScript became a fully independent language with its own specification called **ECMAScript**, and now it has no relation to Java at all.

## Placement

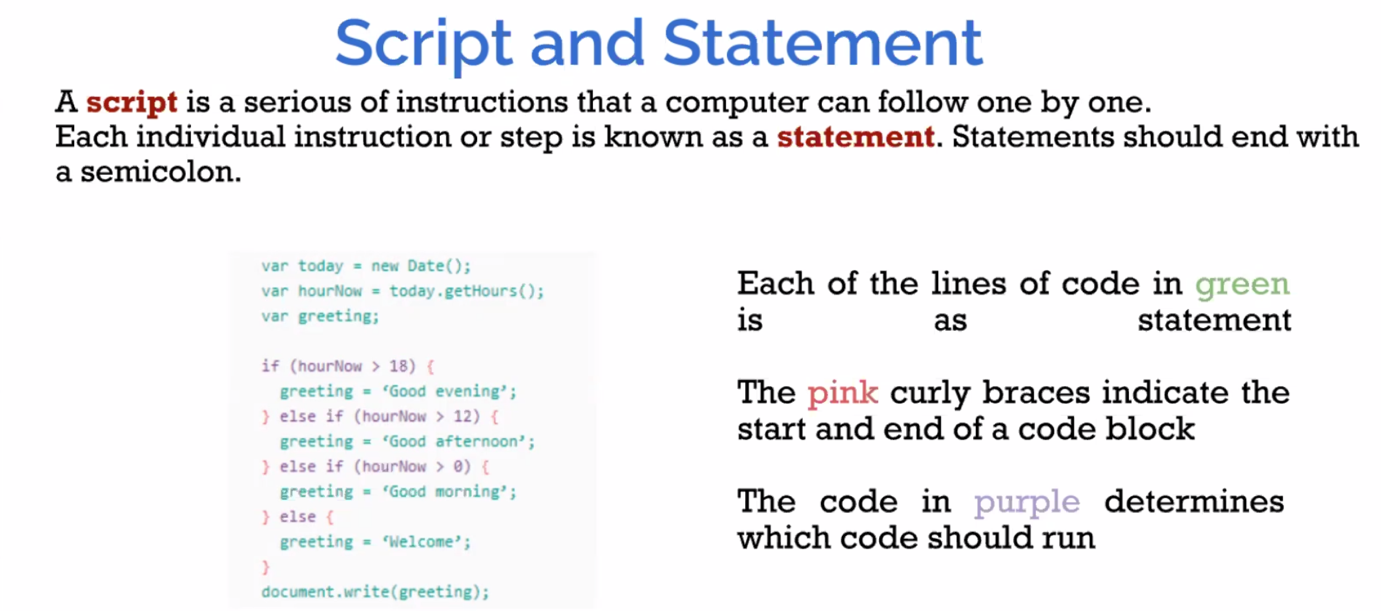
**➤** A **script** is a serious of instructions that a computer can follow one by one.

**➤** Each individual instruction or step is known as a **statement**.  **Statements** should end with a **semicolon**.

**➤** In **HTML**, **JavaScript** code must be inserted between <script> and </script> tags.

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

**➤** **Script** tags can be inserted in the **HTML** page's <body>, or <head> sections, or in both.



## External JavaScript

**➤** **Scripts** can be located in **external files**.

**➤** External **scripts** are useful when the same code is used on many different web pages.

**➤** To include an external **JavaScript** file in your **HTML** code, it is necessary to put the name of the **script** file in the **src** attribute of a **<script>** tag.

**➤** An external **script** reference can be located in the **<head>** or **<body>** section of an **HTML** page.

**➤** The file extension of **JavaScript** files is **.js**.

<script src = "myScript.js" ></script>

## Output

There are different ways to **"display"** data in **JavaScript**.

**➤** document.write().  
**➤** window.alert().  
**➤** console.log().

## Comments

**●** a single line comments after a double slash //  
**●** and multi-line comments between /\* ....... \*/

## ECMAScript 2015 (ES6)

**➤** **ECMAScript (ES)** is a scripting language specification intended to standardize **JavaScript**.

**➤** **ECMAScript 2015**, which is also known as **ECMAScript 6** and **ES6**, is the sixth edition of the **ECMAScript Language Specification** standard.

**➤** **ES6** provides a new set of features and fixes to **JavaScript**.

**★** We’re going to dive into what' difference between ES6 and before its.

**⚡ Commonly Asked Questions:**

* How do I get JavaScript?
* Where can I download JavaScript?
* Is JavaScript Free?

You don't have to get or download JavaScript.  
JavaScript is already running in your browser on your computer, on your tablet, and on your smart-phone.  
Free to use for everyone.

# JS Variables

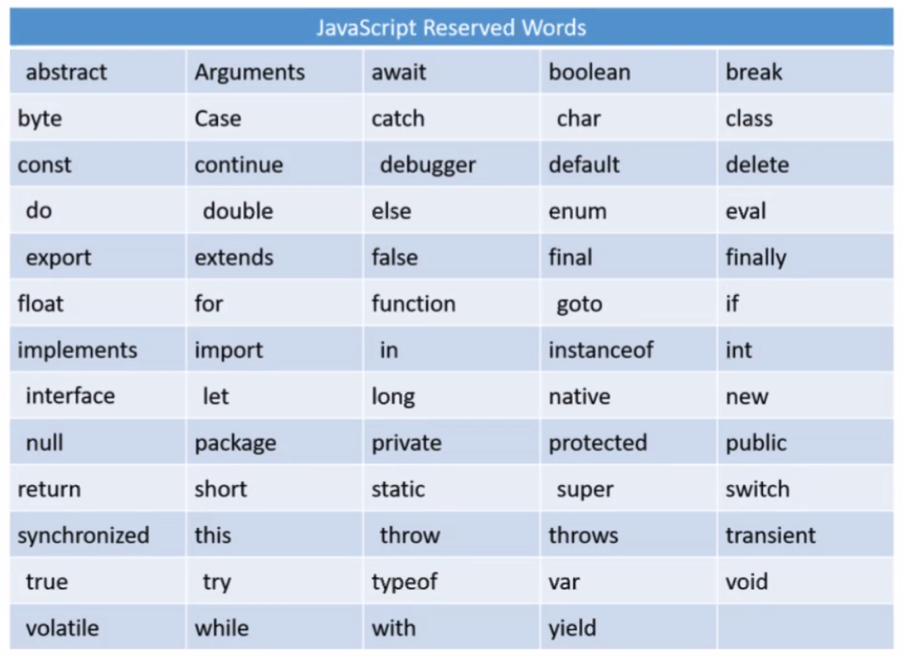
## Naming Rules

The general rules for defining names for **variables**:

**➤** Names can composed of letters, digits, underscores, and dollar signs.  
**➤** Numbers are not allowed as the first character.  
**➤** The first character must be;  
   **●**a letter  
   **●**an underscore ( **\_** )  
   **●**a dollar sign (**$**)

**➤** JavaScript names must not contain spaces, mathematical or logical operators.

**➤** Reserved words cannot be used as names



## JavaScript let and const

**➤** Before **ES6** we used to define a variable using the **var** keyboard.

**➤** **let** and **const** keyboards are added to **JavaScript** with **ES6**.

### let

**➤** The let statement enables you to declare a variable with **block scope**.

**➤** Scope is the fundamental concept that defines a variable's visibility in all programming languages.

Example

var a = 10;

{

let b = 3;

}

console.log ("a = " + a);

console.log ("b = " + b); //generates an error

Output:

a = 10  
ReferenceError: b is not defined

### **const**

**const** variables are similar to **let** variables, except that **const** variables are **immutable.** They are not allowed to be reassigned.

Example

const x = 5;

x = 7; //generates an error

Output:

TypeError: Assignment to constant variable

# JS Data Types

## Data Types

**➤** JavaScript variables can hold numerous data types, such as numbers, strings, booleans, and more.

**➤** It is required to declare a variable for creating.

**➤** In JavaScript, we can declare a variable with the var keyword.

var myNumber;

**➤** The latest ECMAScript standard defines [nine types:](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures)

**➤** Six Data Types that are primitives, checked by typeof operator:

* **Undefined** : typeof instance === "undefined"
* **Boolean** : typeof instance === "boolean"
* **Number**  : typeof instance === "number"
* **String**  : typeof instance === "string"
* **BigInt** : typeof instance === "bigint"
* **Symbol**  : typeof instance === "symbol"

**➤** **null** : typeof instance === "object". Special primitive type having additional usage for its value: if object is not inherited, then null is shown;

**➤** **Object** : typeof instance === "object". Special non-data but structural type for any constructed object instance also used as data structures: new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date and almost everything made with new keyword;

**➤** **Function** non data structure, though it also answers for typeof operator: typeof instance === "function".

This answer is done as a special shorthand for Functions, though every Function constructor is derived from Object constructor.

|  |  |
| --- | --- |
| **console.log ("1. " + typeof 0);**  **console.log ("2. " + typeof (3,14));**  **console.log ("3. " + typeof "hello");**  **console.log ("4. " + typeof (7+8));**  **console.log ("5. " + typeof "Oliver");**  **console.log ("6. " + typeof "");**  **console.log ("7. " + typeof true);**  **console.log ("8. " + typeof (3>=2));**  **var a; // Value is undefined.**  **console.log ("9. " + typeof a);** | 1. **number** 2. **number** 3. **string** 4. **number** 5. **string** 6. **string** 7. **boolean** 8. **Boolean** 9. **undefined** |

### Numbers

**➤** Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc. JavaScript has only one type of **number**.

**➤** **Numbers** can be written with decimals.

var x = 7;

var y = 33.33;

**Precision**

**➤** Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

var x = 999999999999999;

var y = 9999999999999999;

console.log(x); // output = 999999999999999

console.log(y); // output = 10000000000000000

**➤** The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

var x = 0.2 + 0.1;

console.log(x); // output = 0.30000000000000004

**➤** To solve the problem above, it helps to multiply and divide:

var x = 0.2 + 0.1;

console.log(x); // output = 0.30000000000000004

var y = (0.2\*10 + 0.1\*10) / 10;

console.log(y); // output = 0.3

### BigInt

**➤** The **BigInt** type is a numeric primitive in JavaScript that can represent integers with arbitrary precision. With BigInts, you can safely store and operate on large integers even beyond the **safe integer** limit for Numbers.

**➤** A **BigInt** is created by appending n to the end of an integer or by calling the constructor.

**➤** You can obtain the safest value that can be incremented with Numbers by using the constant **Number.MAX\_SAFE\_INTEGER**. With the introduction of BigInts, you can operate with numbers beyond the **Number.MAX\_SAFE\_INTEGER**.

> const x = 2n \*\* 53n;

// 9007199254740992n

> const y = x + 1n;

// 9007199254740993n

**➤** You can use the operators +, \*, -, \*\*, and % with BigInts—just like with Numbers. A BigInt is not strictly equal to a Number, but it is loosely so.

**➤** A **BigInt** behaves like a Number in cases where it is converted to Boolean: if, ||, &&, Boolean, !.

**➤** **BigInts** cannot be operated on interchangeably with Numbers. Instead, a **TypeError** will be thrown.

### Symbol

**➤** **Symbol** is a primitive value.

**➤** A value having the data type Symbol can be referred to as a "Symbol value". In a JavaScript runtime environment, a symbol value is created by invoking the function Symbol, which dynamically produces an anonymous, unique value. A symbol may be used as an object property.

**➤** **Symbol** can have an optional description, but for debugging purposes only.

**➤** A Symbol value represents a unique identifier. For example:

// Here are two symbols with the same description:

let Sym1 = Symbol("Sym")

let Sym2 = Symbol("Sym")

console.log(Sym1 === Sym2) // returns "false"

**➤** **Symbol** type is a new feature in ECMAScript 2015. There is no ECMAScript 5 equivalent for Symbol.

**➤** In some programming languages, the **symbol** data type is referred to as an "**atom**."

### Strings

**➤** Scripts are defined as the sequence of characters.  
**➤** In **JavaScript**, strings are used for storing and manipulating text.  
**➤** A string can be any content that shows up inside **quotes**. We can use **single** or **double-quotes**.

var myName = "John";

var myAge = '36';

**➤** We can use quotes inside a string unless they don't match the quotes encompassing the string.

var text1 = "I'm John";

var text2 = 'She said, "Go ahead"'; // output : She said, "Go ahead"

**➤** If we need to use quotes inside the string, we can use

**backslash ( \)** escape character.

var text = "He said, \"I am a new programmer.\"";

// output : He said, "I am a new programmer."

**Breaking Long Code Lines**

**➤** For best readability, programmers often like to avoid code lines longer than 80 characters.  
**➤** If a JavaScript statement does not fit on one line, the best place to break it is after an operator:

document.getElementById("demo").innerHTML = // Break a code line after

"Hello Dolly!"; // an operator or comma

### Null

**➤** The special **null** value does not belong to any of the types described above.

**➤** It forms a separate type of its own which contains only the null value:

let age = null;

**➤** In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.

**➤** It’s just a special value that represents “nothing”, “empty” or “value unknown”.

**➤** The code above states that **age** is unknown.

### Undefined

**➤** The special value **undefined** also stands apart. It makes a type of its own, just like null.

**➤** The meaning of **undefined** is “value is not assigned”.

**➤** If a variable is declared, but not assigned, then its value is **undefined**:

let age;

console.log(age); // output: "undefined"

### Booleans

**➤** A **JavaScript Boolean** can have one of two values, either true or false.

**➤** **Boolean** data type is useful when we need to have one of two values, such as true/false, yes/no, etc.

**The Boolean() Function**

**➤** You can use the Boolean() function to find out if an expression (or a variable) is true:

Boolean (3 > 2); //return true

Boolean (2 < 3); //return true

Boolean (2 > 3); //return false

Boolean (2 == 3); //return false

**➤** **The Boolean value of everything with a "value" is true.**

Boolean("Hello World");

Boolean(5);

Boolean(-5);

Boolean("false");

Boolean(3.14 + 8); //All of these values return true

**➤** The Boolean value of everything without a "value" is false.

**➤** The Boolean value of 0, null, empty string, undefined, is false.

Boolean("");

Boolean(0);

Boolean(-0);

Boolean(null);

var x;

Boolean(x); //All of these values return false

### Objects

**➤** **Objects** in JavaScript, just as in many other programming languages, can be compared to objects in real life. The concept of objects in JavaScript can be understood with real life, tangible objects.

**➤** In JavaScript, an **object** is a standalone entity, with properties and type. Compare it with a cup, for example. A cup is an object, with properties. A cup has a color, a design, weight, a material it is made of, etc. In the same way, JavaScript objects can have properties, which define their characteristics.

**Object properties**

**➤** A **JavaScript object** has properties associated with it. A property of an object can be explained as a variable that is attached to the object.

var myCar = new Object();

myCar.make = 'Ford';

myCar.model = 'Mustang';

myCar.year = 1969;

**➤** The above example could also be written using an object initializer, which is a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}):

var myCar = {

make: 'Ford',

model: 'Mustang',

year: 1969

};

**➤** Properties of JavaScript objects can also be accessed or set using a bracket notation

myCar['make'] = 'Ford';

myCar.model = "Mustang";

# JS Strings

## String Methods and Properties

**➤** Primitive values, like "Edward Benedict", cannot have properties or **methods** (because they are not objects).

**➤** But with **JavaScript**, **methods** and **properties** are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

### length Property

**➤** length property returns the count of the total number of characters.

**➤** The length of an empty string is 0

var x = "Clarusway";

var y = "";

console.log (x.length); //output: 9

console.log (y.length); //output: 0

### concat() Method

**➤** The concat() method is used to combine two or more strings.

**➤** This method **does not modify** the current strings but returns a new string that contains the joined strings text.

var s1 = "Hello ";

var s2 = "World!";

var s3 = s1.concat(s2);

console.log(s3) ; //output: Hello World!

**➤** The concat() method can be used instead of the **plus**(+) operator.

### charAt() Method

**➤** The charAt() method returns the char value at the specified index in a string.

var x = "Clarusway";

console.log (x.charAt(0)); //output: C

console.log (x.charAt(3) ); //output: r

**❗ Warning :** The index of the first character is 0 and the index of the last character is string.length-1.

### includes() Method

**➤** The includes() method specifies whether a string includes the characters of a specified string.

**➤** This method returns true if the characters are in the string, and if not false.

var s = "Hello John, welcome to Clarusway.";

var n = s.includes("welcome");

console.log (n); // output: true

**❗ Warning :** The includes() method is **case sensitive**.

### indexOf() and lastIIndexOf() Method

**➤** The indexOf() method returns the **index** of (the position of) the **first** occurrence of a specified text in a string:

**➤** This method returns **-1** if the value is **not found**.

var s = "Hello John, welcome to Clarusway.";

var n = s.indexOf("welcome");

console.log (n); // output: 12

**➤** You can find second or other letters in the string with second parameter of indexOf() method. Find the first occurrence of the letter "e" in a string, **starting the search at position 5**:

var s = "Hello John, welcome to Clarusway.";

var n = s.indexOf("e", 5);

console.log (n); //output : 13

### lastIndexOf() Method

**➤** The lastIndexOf() method returns the **index** of the **last** occurrence of a specified text in a string:

var s = "Hello John, welcome to Clarusway. How are you John?";

var n = s.lastIndexOf("John");

console.log (n); //output : 46

### replace() Method

**➤** The replace() method looks for a string for a **given value** and returns a **new string** to **replace** the specified values.

var s = "Hello John, welcome to Clarusway.";

var rep = s.replace("Clarusway", "Course")

console.log (rep); output : Hello John, welcome to Course.

**❗ Warning :**  
The replace() method does not change the string it is called on. It returns a **new string**.  
  
By default, the replace() method replaces **only** the **first match** and the replace() method is **case sensitive**.

**➤** To replace **case insensitive**, use a **regular expression** with an /i flag (insensitive):

var str = "Hello John, welcome to Clarusway.";

var rep = str.replace(/JOHN/i, "Edward")

console.log (rep); //output: Hello Edward, welcome to Clarusway.

### search() Method

**➤** The search() method searches a string for a given value and returns the **position** of the match.

**➤** This method returns -1 if the value is not found.

var s = "Hello John, welcome to Clarusway.";

console.log (s.search("Clarusway"));// output: 23

**❗ Warning :**  
The two methods, indexOf() and search(), are **equal**? **NOT** equal.

* The search() method cannot take a second start position argument.
* The indexOf() method cannot take powerful search values (regular expressions).

### slice() Method

**➤** The slice() method **extracts a section** of a string and returns it as a **new string**.

**➤** We can use a **negative** number to select from the **end** of the string.

|  |  |
| --- | --- |
| **var s = "Hello John, welcome to Clarusway.";**  **console.log (s.slice(0, 5));**  **console.log (s.slice(6, 10));**  **console.log (s.slice(12));**  **console.log (s.slice(12, -10));** | **Hello John welcome to Clarusway. welcome to** |

**➤** If you dont use the **second parameter**, the method will slice out the **rest** of the string:

|  |  |
| --- | --- |
| **var s = "Hello John, welcome to Clarusway.";**  **var rest = s.slice(12);**  **console.log(rest)** | **welcome to Clarusway.** |

### split() Method

**➤** The split() method **divides** a string into an **array** of substrings, and returns the **new array**.

**➤** The split() method **does not alter** the original string.

var s = "Hello John, welcome to Clarusway."

var arr = s.split(" ");

console.log (arr); // output : ["Hello", "John,", "welcome", "to", "Clarusway."]

**➤** If the **separator** is omitted, the returned array will contain the whole string in **index [0]**.

**➤** If the separator is "", the returned **array** will be an array of **single characters**.

### substr() Method

**➤** The substr() method returns the **parts** of a string, **beginning** at the character at the specified position and a **number** of characters after it.

**➤** The substr() method **does not alter** the original string.

var s = "Hello John, welcome to Clarusway."

console.log (s.substr(23, 9)) ;// output: Clarusway

### substring() Method

**➤** The substring() method returns the **parts** of a string between "start" and "end", **not including "end"** itself.

**➤** The substring() method **does not alter** the original string.

var s = "Welcome to Clarusway."

console.log (s.substring(5, 10)); // output : me to

console.log (s.substring(5, 1)); // output : elco

**➤** If you omit the second parameter, substring() will slice out the **rest** of the string.

Differences between slice() and substring()

|  |  |
| --- | --- |
| **What they have in common:**   1. If start equals stop: returns an empty string 2. If stop is omitted: extracts characters to the end of the string   If either argument is greater than the string's length, the string's length will be used instead. | |
| **Distinctions of** [substring()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/substring)**:**   1. If start > stop, then substring will swap those 2 arguments. 2. If either argument is negative or is NaN, it is treated as if it were 0. | **Distinctions of** [slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice)**:**   1. If start > stop, slice() will return the empty string. ("") 2. If start is negative: sets char from the end of string, exactly like substr() in Firefox. This behavior is observed in both Firefox and IE. |

### toLowerCase() Method

**➤** The toLowerCase() method transforms a string to **lowercase** letters.

**➤** The toLowerCase() method **does not alter** the original string.

var s = "WELCOME TO CLARUSWAY";

console.log(s.toLowerCase()); // output : welcome to clarusway

### toUpperCase() Method

**➤** The toUpperCase() method transforms a string to **uppercase** letters.

**➤** The toUpperCase() method **does not alter** the original string.

var s = "Welcome to Clarusway"

console.log(s.toUpperCase());// output : WELCOME TO CLARUSWAY

### trim() Method

**➤** The trim() method **eliminates** whitespace from both sides of a string.

**➤** The trim() method **does not alter** the original string.

var s = " Welcome to Clarusway "

console.log (s. trim()); // output : Welcome to Clarusway

## Template literals ``[[1]](#footnote-1)

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.

They were called "template strings" in prior editions of the ES2015 specification.

`string text`

`string text line 1

string text line 2`

`string text ${*expression*} string text`

*tag*`string text ${*expression*} string text`

Template literals can contain placeholders. These are indicated by the dollar sign and curly braces (${*expression*}). The expressions in the placeholders and the text between the backticks (` `) get passed to a function.

[Click for more info](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals)

# Operators

**➤** Let's take a simple 3 + 2 phrase equals 5. Number 3 and 2 are **operands** and ‘+’ is the **operator**.

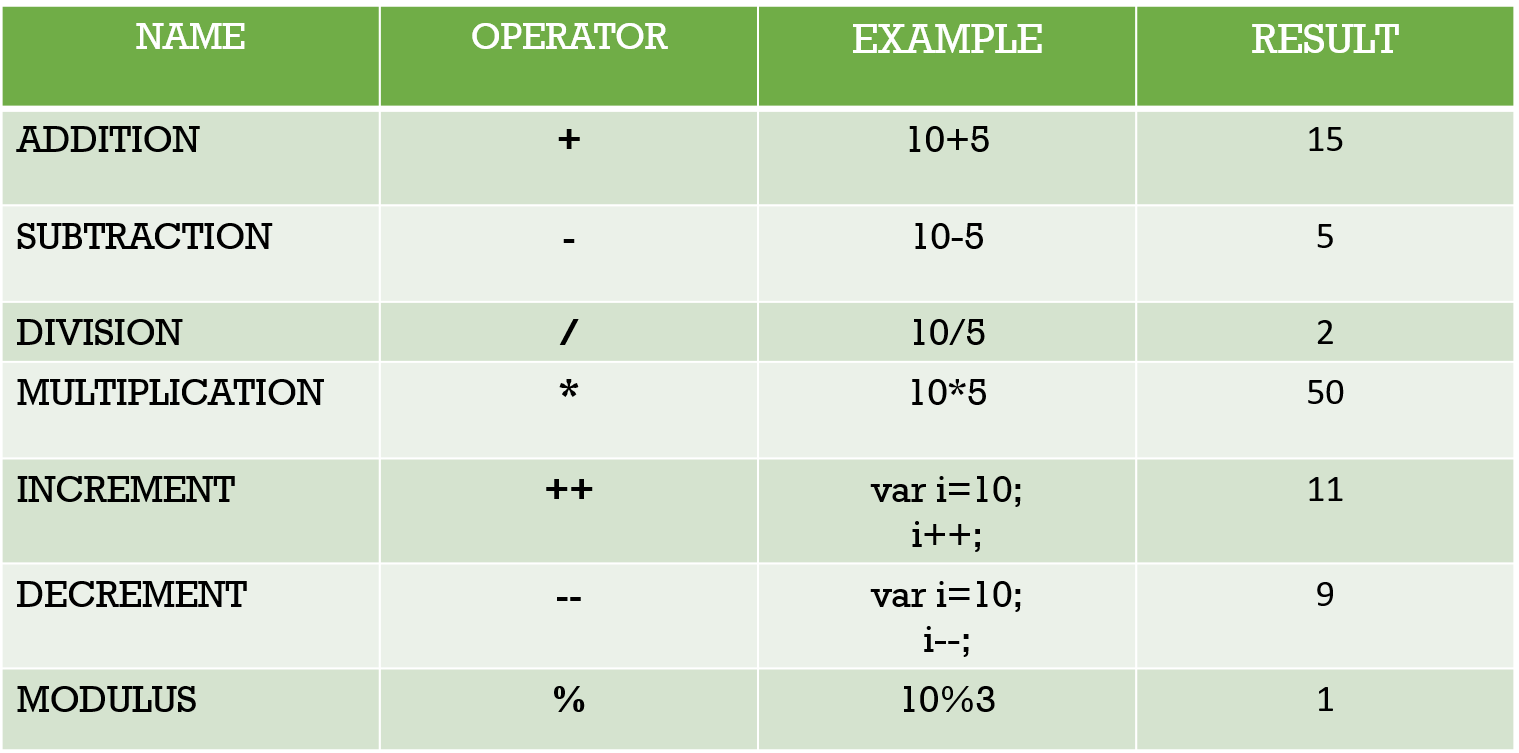
**➤** Expressions rely on **operators** to create a single value from one or more values.

**➤** JavaScript supports the **operators** of the following types.

**●** Arithmetic Operators  
**●** Assignment Operators  
**●** Comparison Operators  
**●** Logical Operators

## Arithmetic Operators

**➤** **Arithmetic operators** execute arithmetic functions on numbers (literals or variables).

****

var a = 20;

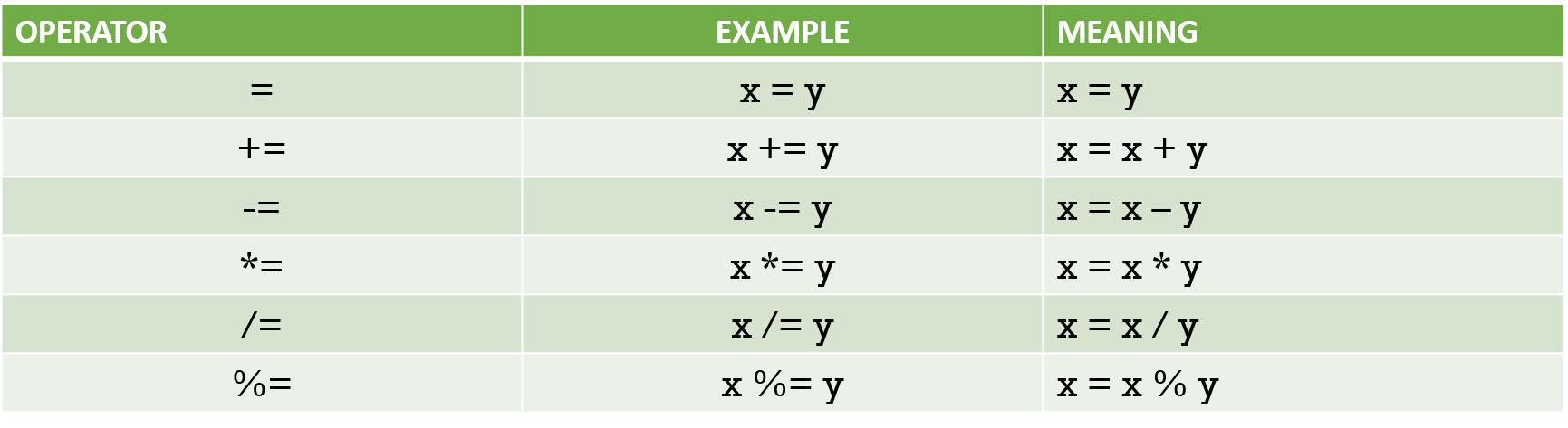
var b = 3;

var c = a \* b;

console.log (c); //output: 60

## Assignment Operators

**➤** **Assignment operators** assign values to JavaScript variables.



var a = 20;

a -= 5;

console.log (a); // 15

var a = 20;

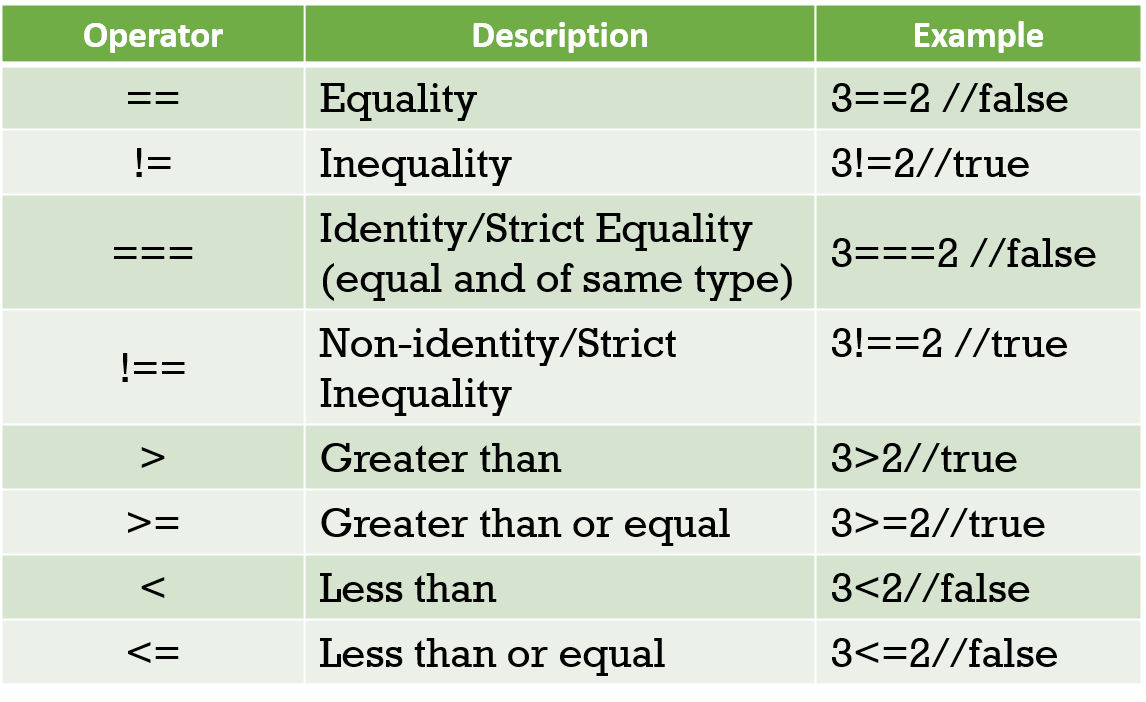
a %= 3;

console.log (a); //2

## Comparison Operators

**➤** **Comparison operators** are used to determine equality or difference between variables or values in logical statements.

**➤** All comparison operators return **Boolean** (true or false).



var a = "10";

console.log (a == 10);

// true

var a = "10";

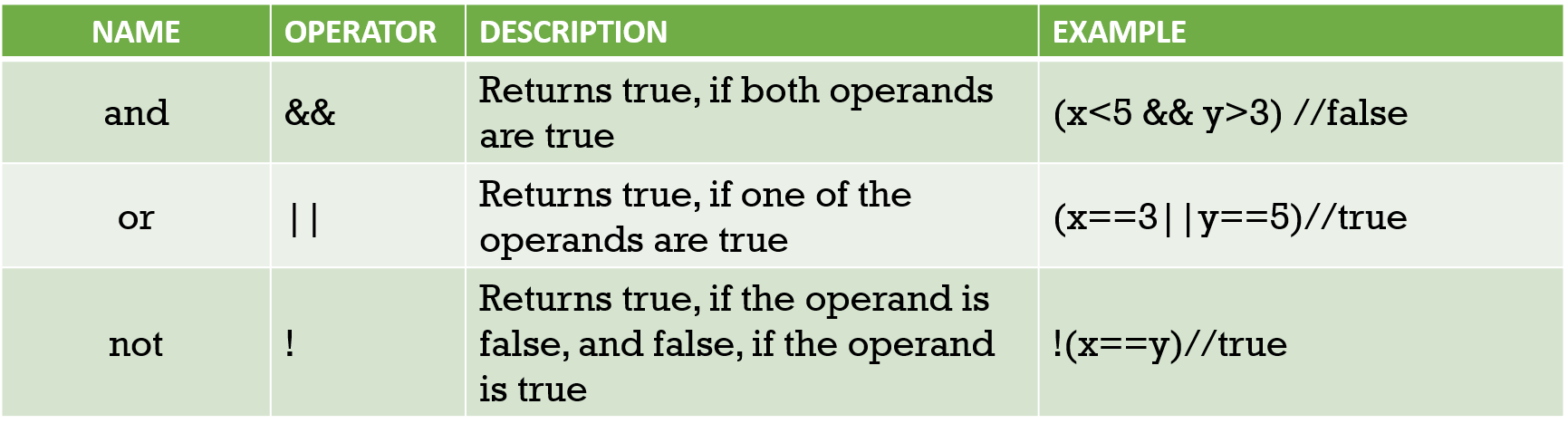
console.log (a === 10);

// false

## Logical Operators

**➤** Logical operators, also known as **Boolean Operators**, are used to determine the logic between variables or values and return true or false.

**➤** Seeing as x = 3 and y = 2, logical operators are explained in the table below:



var a = 10;

var b= 5;

console.log (a>=b && b<=20 );

// true

var a = 10;

var b= 5;

console.log (a==b || b<=20 );

// true

var a = 10;

var b= 5;

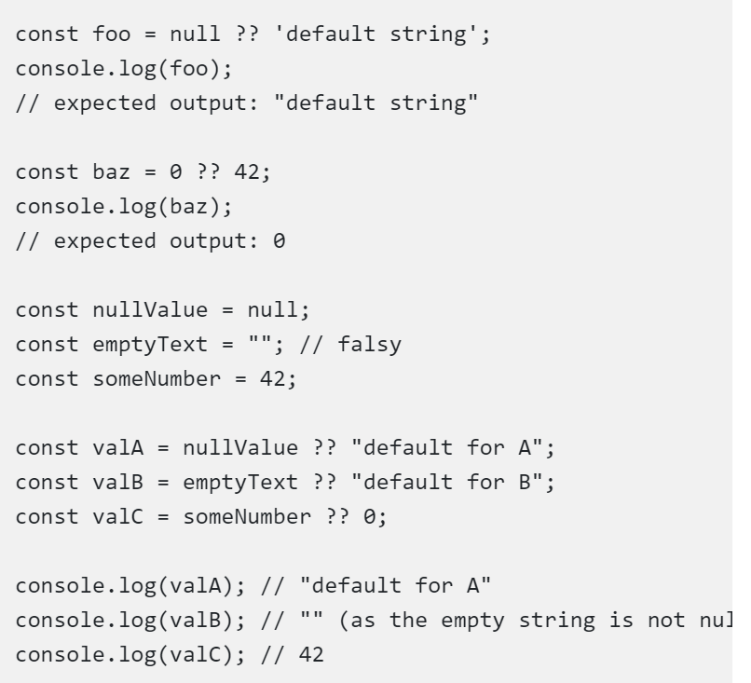
console.log (!(a>=b ));

// false

### Nullish coalescing operator (??)

**➤** The **nullish coalescing operator (??)** is a logical operator that returns its right-hand side operand when its left-hand side operand is **null** or **undefined**, and otherwise returns its left-hand side operand.

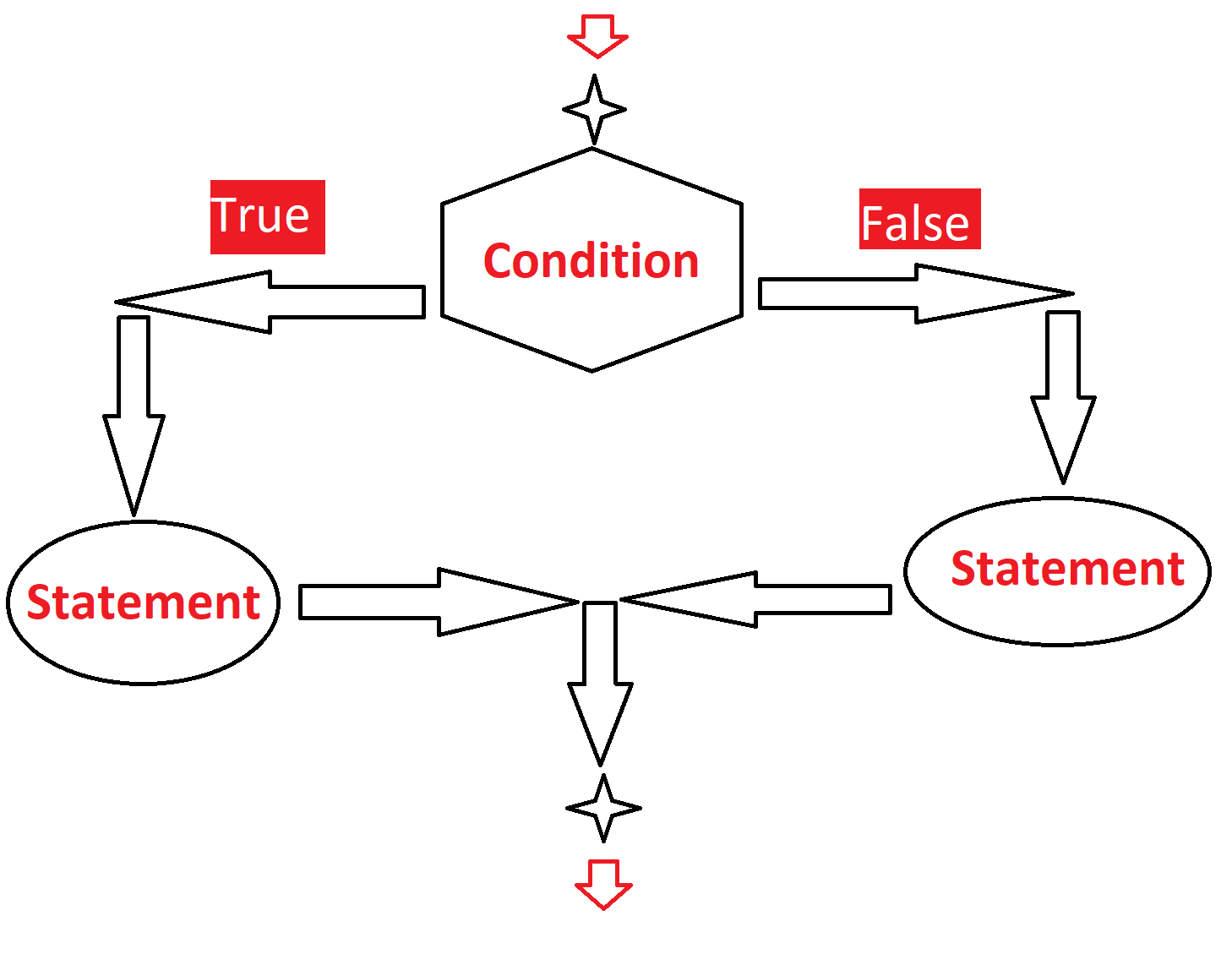
**➤** Contrary to the logical **OR** (**||**) operator, the left operand is returned if it is a falsy value that is not **null** or **undefined**. In other words, if you use **||** to provide some default value to another variable **foo**, you may encounter unexpected behaviors if you consider some falsy values as usable (eg. **''** or **0**). See below for more examples.



# JS Conditionals

## Conditional Statements

**➤** Based on different conditions, **conditional statements** are used to decide the **execution flow**. If a condition is true, one action can be performed and you can perform another action if the condition is false.



**➤** You want to perform different actions for different decisions very often when you write code.

**➤** You can use the code's **conditional statements** to do this.

**➤** The following **conditional statements** are available in **JavaScript**:

1. **If Statement:** If a specified condition is true, Use if to specify a code block to be executed
2. **else:** If the same condition is false, Use else to specify a code block to be executed
3. **else if:** If the first condition is false, Use else if to specify a new condition to test
4. **switch:** compare a value with multiple variants.

## If Statement

 If a condition is valid, use the **if statement** to define a block of JavaScript code to execute.

if (condition) {

// Statement (if the condition is true, statement to be execute)

}

var x = 3;

if (x == 3) {

console.log(x); //(the condition is true and statement to be execute)

}

// 3

## Else Statement

**➤** If the condition is false, use the other statement to define a block of code to be executed.

var time = 12;

if (time < 8) {

console.log("Good morning.");

} else {

console.log("Good day.");

}

## if…else if…else Statement

**➤** If the first condition is false, use the other if the statement defines a new condition.

var time = 21.00;

if (time < 8) {

console.log("Good morning.");

} else if (time < 18) {

console.log("Good afternoon.");

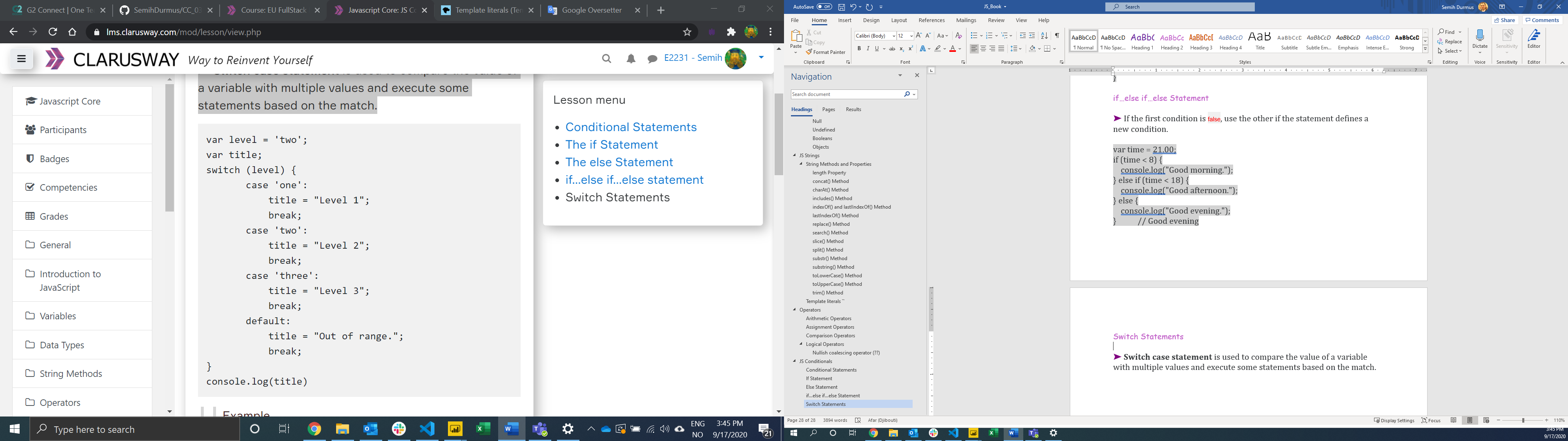
} else {

console.log("Good evening.");

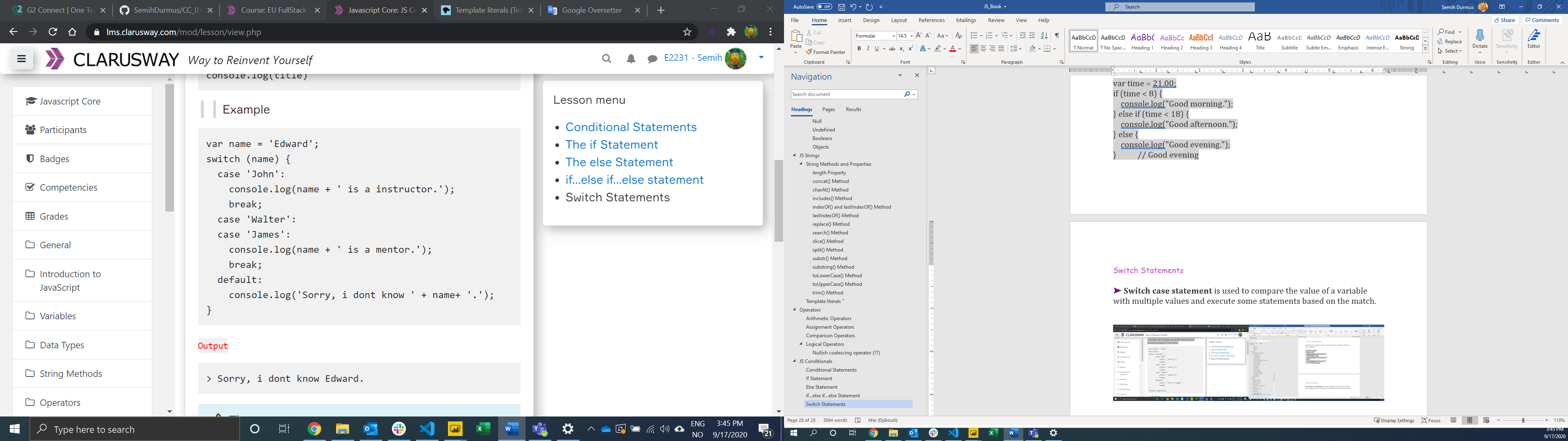
} // Good evening

## Switch Statements

**➤** **Switch case statement** is used to compare the value of a variable with multiple values and execute some statements based on the match.



Level 2



**⚜️ Tips:**

If you omit the break statement, the next case will be executed even if the evaluation does not match the case. But the last default break is **not necessary** to break the last case in a **switch block**. The block breaks (ends) there anyway.

**⚜️ Tips:**

Switch cases use **strict** comparison (===). Because of that, the values must be of the **same type** to match.

1. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template\_literals [↑](#footnote-ref-1)