

# Basic Constructs: Conditional Statements and Iterative Statements

**Relevel**  
by Unacademy



# Topics for the Day



Introduction to programming



Introduction to Javascript



Why use JavaScript



Intro to Mozilla Firefox  
multi-line console

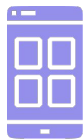


Hello World Program in  
Javascript



`console.log()`

# Topics for the Day



Variables



Operators



Coercion



Data types



Comments

# Introduction to programming

Programming is a way to write tasks to be performed by the computer.

A program can be written in a programming language.

Many programming languages like C, C++, Javascript, Python, Java, etc.

Each language has its own rules, following which we need to write a set of instructions that the computer can perform.

Programming tasks could be as simple as adding two numbers and writing complex algorithms that help rockets land on the moon.

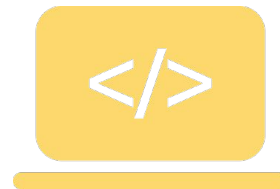


# Introduction to Javascript

Simply put, Javascript is a programming language that helps you write instructions that a computer, to be more specific, a browser can understand.

Javascript was primarily used to be executed on browsers, but then NodeJS was created, which can help execute Javascript on your machine outside the browser.

Javascript is a *single-threaded, synchronous, high-level programming language*.

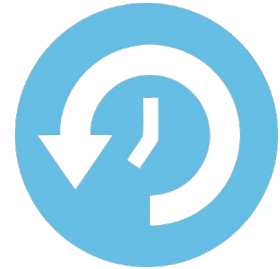


Let's understand the previous statement clearly.

- **Single-threaded** means that a javascript program cannot be broken into sub-programs and execute the sub-programs parallelly. Understand the entire program as rope and a sub-program as a thread of the rope.
- **Synchronous** means that the order of execution will always happen sequentially; the program can't skip some lines of code and come back to it later. But that being said, we can perform asynchronous tasks in javascript using web APIs; we'll learn about this later
- **High-level programming language** means that it is more human-readable and understandable and less machine-understandable.

# History of Javascript

- JavaScript was invented by Brendan Eich in 1995. He was recruited by Netscape to develop a language which can be run in the browser.
- In 1996, Microsoft released their own version of Javascript and it was named as JScript. To prevent this conflict, Netscape went to the European Computer Manufacturers Association, aka ECMA to develop standard Javascript. They standardized Javascript and renamed it as ECMAScript, since it was developed by them.



# Why use Javascript

Why should you learn javascript? Well, there are a few reasons for that.

A program will always be in javascript on the front-end that is the UI of any website. The browser understands only HTML, CSS, and Javascript.

Now, there are languages like GWT(Java), dart, etc., using which we can create UI, but those languages translate the code into javascript, and then the javascript code gets executed on the browser.





So why not write your code in javascript itself, you'll save time that takes to convert any language to javascript and also you'll have more control over the code.

Another reason is that once you learn javascript you can become a full-stack developer without the need to learn any other language.

Also, Javascript is used to create Desktop apps with Electron and React native desktop frameworks.

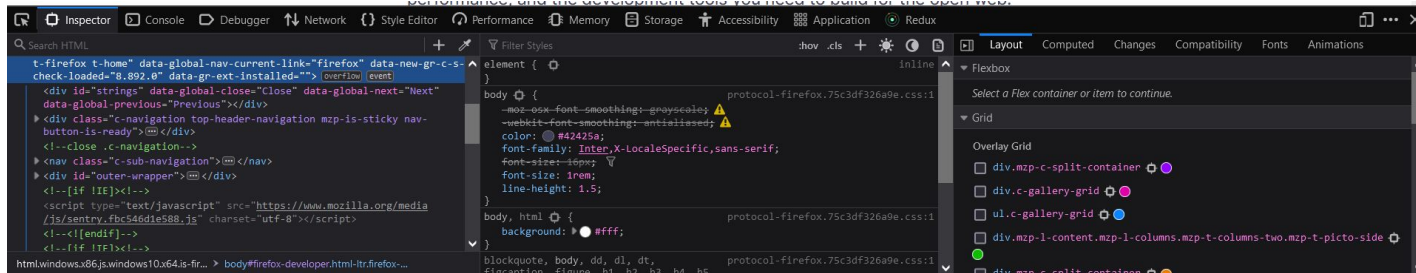
And Javascript is also used to create android and IOS apps using React native.

# Mozilla Firefox multi-line console

Firefox provides a multi-line console where you can write and execute your javascript code without the need for a code editor.

To use it, first download firefox from [here](#) and install it in your system.

Open the Firefox browser and press 'F12' OR right-click and click on inspect. You'll see the dev tools window open like below

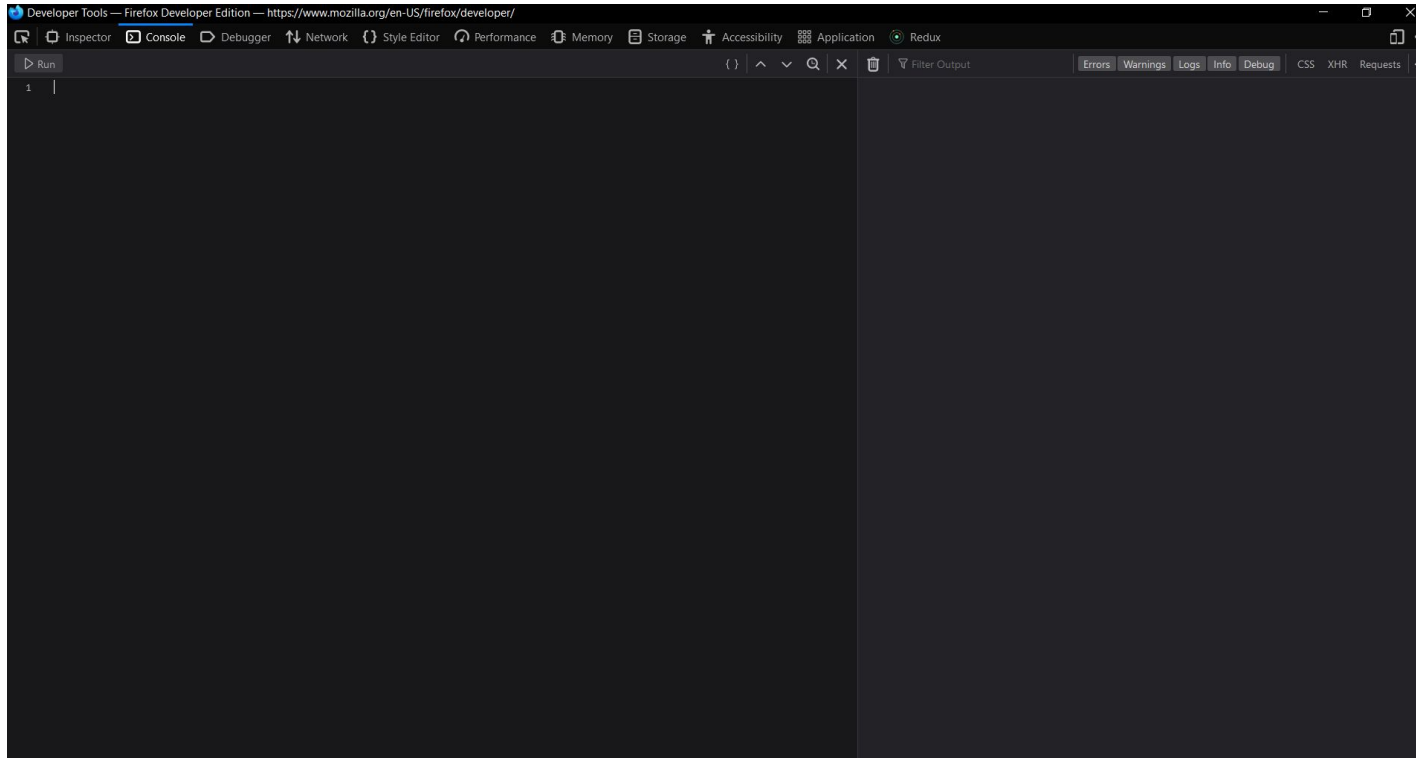


First, click on the 'more options' icon ( top-right corner ) besides the 'close' icon and choose the 'separate window' option. Now your dev-tools window will be open in a separate window.

Select console tab.

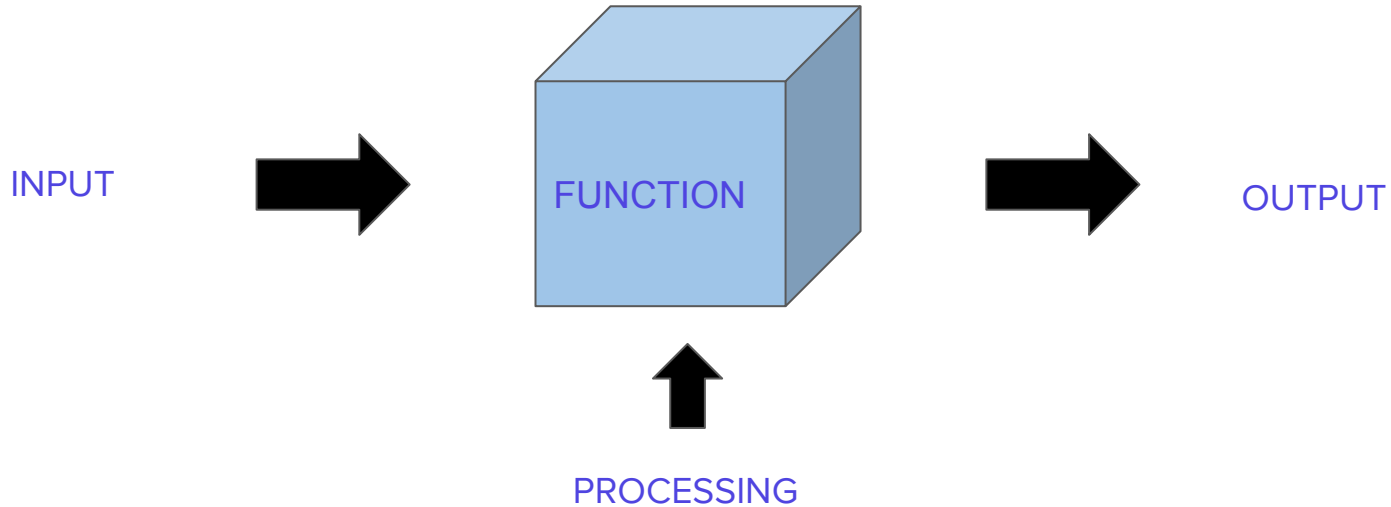
On the left side, we have the code panel and right side we have the console. You can write your javascript code on the code panel and click on 'Run' and the output will be shown on the console on right.

Your window will look like below



# Understanding function

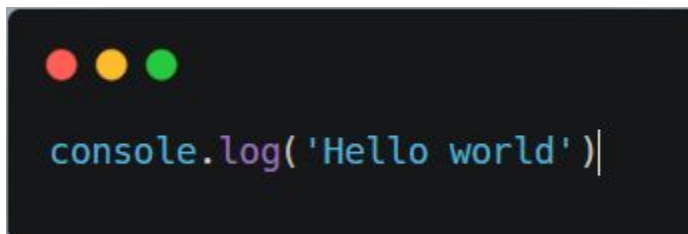
A function is a block of code which can perform some task. It takes some input, applies some logic on it and gives the output.



# Writing an executing first program: Hello World

Usually, when we learn a new programming language we always start by executing a 'hello world' program in it. So, let's do it with Javascript too.

Write the following line of code in firefox code panel



```
console.log('Hello world')|
```

And click on 'Run'

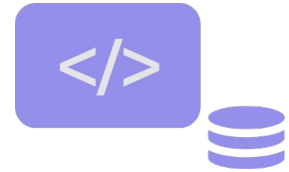
You should see 'Hello world' on your console.

Awesome, you just ran your first code in javascript. Now let's understand the line of code.

'console.log()' is a function that helps you print any data on the browser console or on your terminal window if you are executing the javascript code in nodejs.

# Understanding console.log()

Diving deeper into 'console.log()', 'console' is an object that provides access to the browser's debugging console. And 'log' is a method in console object that will print whatever string you pass as an argument to it. Now, I know there are many terms here that you might not understand like what is an object, what is a method, what do you mean by argument. These you'll understand as and when you proceed with the coming sessions. But for now, just remember what is 'console.log()'






# Variables

In order to store values like numbers, words, objects, etc we need variables.

Variables are placeholders for values.

Every language has specific keywords to declare variables.

In Javascript, you can create variables using any of the following keywords



```
var num1 = 10  
let str1 = 'variable'  
const bool = true
```

When we declare a variable we name it like 'num1', 'str1', 'bool' in the above value. And also we assign a value to it to be stored in the variable. So that we can use it later in our program when needed.

Let's try to understand variables in-depth, when we declare a variable the system memory reserves a memory location and stores the value assigned to it in that memory space. And when we reference the variable name again in the program, the value is fetched from that memory space.

Let's understand the difference between var, let, and const

- When we declare a variable with 'var' keyword, it is function-scoped
- When we declare a variable with 'let' keyword, it is block-scoped
- When we declare a variable with 'const' keyword, it is block-scoped, but we cannot re-assign the value. The value with which we initialise the const variable will remain constant throughout the program.

We'll learn about function-scope and block-scope in later sessions

var	const	let
Globally scoped or function scoped. This means that the scope of variables defined by this keyword can be either in the complete program or inside the complete function where we have defined it.	These variables cannot be redefined again or reassigned again. Once defined, their values will be constant	These variables' scope remains under the block.
<pre>var a; a=5;</pre>	<pre>const a = 5; a = 6; //error: Assignment to constant variable</pre>	<pre>{ var a = 2;   let b = 3;   console.log(a); // this will print 2   console.log(b); // this will print 3   console.log(a); // this will print 2   console.log(b); //this will print undefined</pre>
These can be declared without being initialized	These cannot be declared without being initialized	These can be declared without being initialized

# Comments in Javascript

Comments are basically textual information in your code that is ignored by the compiler and is used to communicate information about the code.

Commenting your code is a good practice for others to understand what your code does.

For single-line comments use `//`

```
// a single-line comment in javascript
```

For multi-line comments use `/*...*/`

```
/* a multi-line  
comment in javascript */
```

# Data types

There are seven primitive data types and an object data type.

1. Number: used to represent numbers including negative, positive and 0
2. String: used to represent textual data
3. Boolean: values are 'true' and 'false'
4. Null: value is 'null', represents the intentional absence of any object value.
5. Undefined: value is 'undefined', represents value as uninitialised
6. BigInt: The BigInt type is a numeric primitive in JavaScript that can represent integers with arbitrary precision.
7. Symbol: A Symbol is a **unique** and **immutable** primitive value and may be used as the key of an Object property

```
let a = 10
console.log(typeof a) // returns number
let b = 'avengers'
console.log(typeof b) // returns string
let c = true
console.log(typeof c) // returns boolean
let d = null
console.log(typeof d) // returns 'object' as null is a type of object
let e = undefined
console.log(typeof e) // returns undefined
let f = BigInt(9007199254740991)
console.log(typeof f) // returns bigint
let g = Symbol('avengers')
console.log(typeof g) // returns symbol
```

## OUTPUT:

number

string

boolean

object

undefined

bigint

symbol



- An object is a special data type that contains a series of key-value pairs.
- An array is also a special type of object that contains a list of values; it can contain values of all data types, including objects

```
let a = {  
  num: 10,  
  str: 'avengers',  
  isOpen: false  
}  
console.log(typeof a) // returns object  
let b = [1, 'assemble', true, {a: 1, b: 3}]  
console.log(typeof b) // returns object as array is a type of object in javascript
```

OUTPUT:

```
object
```

```
object
```

# Numbers

Numbers are collections of different digits. They can be integers or decimal numbers.

There are different types of numbers like integers, decimal numbers, exponential numbers, special numbers like Infinity, NaN and so on.

Example –

```
1. var num = 123; //Integer
```

```
2. var num = 212.13456
```

```
    console.log(num.toExponential(4)); //toExponential function is used to write    exponential numbers.
```

```
    Output -> 2.13e+0
```

We can directly write numbers in exponential form too. For example

```
var num = 2.13e+10;
```

# Numbers

3. Special numbers like Infinity and NaN are kind of error values.

## NaN

When any operation got failed or parsing failed, we get NaN as error

```
Number('Hello')
```

Output -> NaN

In the above code, Number() is a function to get the number value of argument passed. Here we passed 'Hello' which is a string and cannot be parsed. hence we got NaN as output

## Infinity

When number value is too large or it is divided by zero, we get Infinity as output

```
a = 3/0  
console.log(a)  
Output -> Infinity
```

```
a = 3/ -0  
console.log(a)  
Output -> -Infinity
```

# Boolean

Boolean can be 2 values either true or false. Whenever we want to store a value whose value can be either true or false, we can use Booleans datatype to store it.

Example – let x = true;

# Null and Undefined –

“Null” and “undefined” refers to null value or when we don’t assign any value to it. When we do not give value to a variable, it denotes undefined. When we give an empty value to the variable, it will be considered null.

## Example –

```
var a = null;
```

```
var b;
```

```
console.log(a) // print null
```

```
console.log(b) // print undefined
```

# String –

A string is a collection of alphanumeric characters and symbols. To store different words, letters, and sentences, we use strings. In JavaScript, we can either use single or double quotes to define it. We can also use backticks to define strings.

Example –

```
var str = "Hello World" // double quotes
```

```
var str = 'Hello World' // single quotes
```

```
var str = `Hello World` //backticks
```

```
var multilineStr= `Say hello
```

```
to multi-line
```

```
string`;
```

Hello, World datatype is a string in the above example.

# Operators

When we want to execute any operation like addition, subtraction, division, multiplication, etc in a programming language, we have to use 'operators.'

There are various types of operators in javascript,

Increment/Decrement operators: a++, --a,

Unary operators: typeof, +, !

Arithmetic operators: +, -, /, \*

Relational operators: >, <, >=, <=

Equality operators: ==, !=, ===, !==

Assignment operators: +=, -=, /=, \*=



# Increment/Decrement operators

The increment operator (++) increments (adds one to) its operand and returns a value.

The decrement operator (--) decrements (subtracts one from) its operand and returns a value.

If used postfix, with operator after operand (for example, x++/x--), the increment/decrement operator increments/decrements and returns the value before incrementing/decrementing.

If used prefix, with operator before operand (for example, ++x/--x), the increment/decrement operator increments/decrements and returns the value after incrementing//decrementing.

```
let a = 10
console.log(a++) // returns and then increments
console.log(a) // incremented value
console.log(a--) // returns and then decrements
console.log(a) // decremented value
console.log(++a) // increments and returns the value
console.log(--a) // decrements and returns the value
```

## OUTPUT:

```
10
11
11
10
11
10
```

# Unary operators

- 'typeof' operator returns the data type of the value
- '+' operator attempts to convert a value into a number if it isn't already a number
- '!' operator converts truthy values to falsy and vice versa

```
let num = 10
let str = 'avengers'

console.log(typeof str) // returns string
console.log(typeof num) // returns number

let num2 = '20'
let str2 = 'assemble'
console.log(+num2) // converts string '20' to number 20
console.log(+str2) // returns NaN, meaning not a number
                  // as it cannot convert 'assemble' to a number

let num3 = 4
let str3 = 'avengers assemble'
let num4 = 0
console.log(!num3) // a number(except 0) is a truthy value so it returns false
console.log(!str3) // a string is a truthy value so it returns false
console.log(!num4) // 0 is a falsy value so it returns true
```

## OUTPUT:

```
string  
number  
20  
NaN  
false  
false  
true
```

# Falsy Values

- The falsy values in javascript are 0, On(BigInt) , null, undefined, NaN, false and ""(empty string). These values are evaluated as false by the javascript engine. Apart from these values, all remaining values are truthy values in javascript.

# Arithmetic operators

These operators perform operations like addition, subtraction, division, multiplication, etc

```
let num1 = 20
let num2 = 5

console.log(num1 + num2) // returns the sum of 20 and 5
console.log(num1 - num2) // returns the value after subtracting 5 from 20
console.log(num1 / num2) // returns the value after dividing 20 with 5
console.log(num1 * num2) // returns the value after multiplying 20 with 5
```

OUTPUT:

```
25
15
4
100
```



# Relational operators

A comparison operator compares its operands and returns a boolean value based on whether the comparison is true.

```
let num1 = 20
let num2 = 5

console.log(num1 > num2) // returns true as 20 is greater than 5
console.log(num1 < num2) // returns false as 20 is not less than 5
console.log(num1 >= 20) // returns true
console.log(num2 <= 5) // returns true
```

OUTPUT:

```
true  
false  
true  
true
```

# Equality operator

- “!=’ operator (Inequality Operator) checks if two values are not equal. It will perform type coercion.  
Example -  
‘1’ != 1 -> return false  
1 != 1 -> return false
- ‘!==’ operator (Strict Inequality Operator) will not perform type coercion before comparing two values.  
Example -  
‘1’ !== 1 -> return true  
1 !== 1 -> return false

# Equality operator

- '==' operator checks abstract equality i.e. it will perform required type coercion before applying equality comparison. While '===' operator checks strict equality i.e. it will not perform any type coercion. If two values are not equal, then it will simply return false.
- **Example -**  
9 == "9" -> return true (string "9" is converted to number 9)  
9 === "9" -> return false ( no type coercion takes place here)

# Equality operator

- '==' and '!=' checks if two values are the same or not
- '===' and '!== are strict equality operator checks if two values are the same including the data types

```
let num1 = 20
let num2 = '5' // here '5' is a string

console.log(num1 == num2) // returns false as 20 is not equal to 5
console.log(num1 != num2) // returns true
console.log(num2 == 5) // returns true
console.log(num2 === 5) // returns false as num2 is string but 5 is number
```

# OUTPUT

```
false
```

```
true
```

```
true
```

```
false
```

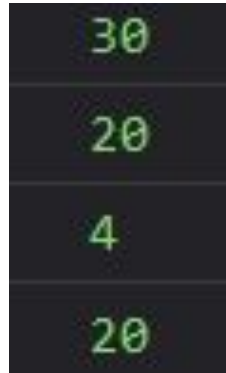
# Assignment operators

'+=', '-=', '/=', '\*=' these operators perform the respective arithmetic operations and then assign the resultant values.

```
let num1 = 20

console.log(num1 += 10) // adds 10 to 20 so num1 becomes 30
console.log(num1 -= 10) // subtracts 10 from 30 so num1 becomes 20
console.log(num1 /= 5) // divides 20 by 5 so num1 becomes 4
console.log(num1 *= 5) // multiplies 4 with 5 so num1 becomes 20
```

OUTPUT:



30  
20  
4  
20



# Type Coercion

Type coercion is the automatic or implicit conversion of values from one data type to another (such as strings to numbers).

```
let num1 = 20
let num2 = '10'
let num3 = 'avengers'

console.log(num1 + num2) // converts 20 to string and concatenates it with 10
console.log(num1 + num3) // converts 20 to string and concatenates it with avengers
```

## OUTPUT:

```
2010
20avengers
```

# Homework

1. Create a basic calculator program using Javascript
2. Use different arithmetic operators and variables to show working of calculator



# Upcoming Class Teaser

conditional Statements

switch case

ternary operators

comparison between the different conditional constructs

loops/iterative Statements

for, while, do-while loops

comparison between the different iterative constructs



**Thank you**