JavaScript

Scripting Language and Basics of JavaScript

Introduction

All scripting languages are programming languages. The scripting language is basically a language where ***instructions are written for a run time environment***. They do not require the compilation step and are rather interpreted. It brings new functions to applications and glue complex system together. A scripting language is a programming language designed for integrating and communicating with other programming languages.

In this step by step JavaScript challenge, you will learn JavaScript, the most popular programming language in the history of mankind. JavaScript is used ***to add interactivity to websites, to develop mobile apps, desktop applications, games*** and nowadays JavaScript can be used for ***machine learning*** and ***AI***.

JavaScript (JS) has increased in popularity in recent years and has been the leading programming language for six consecutive years.

Requirements

1. A computer
2. Internet
3. A browser
4. A code editor

Basics of JavaScript

Console

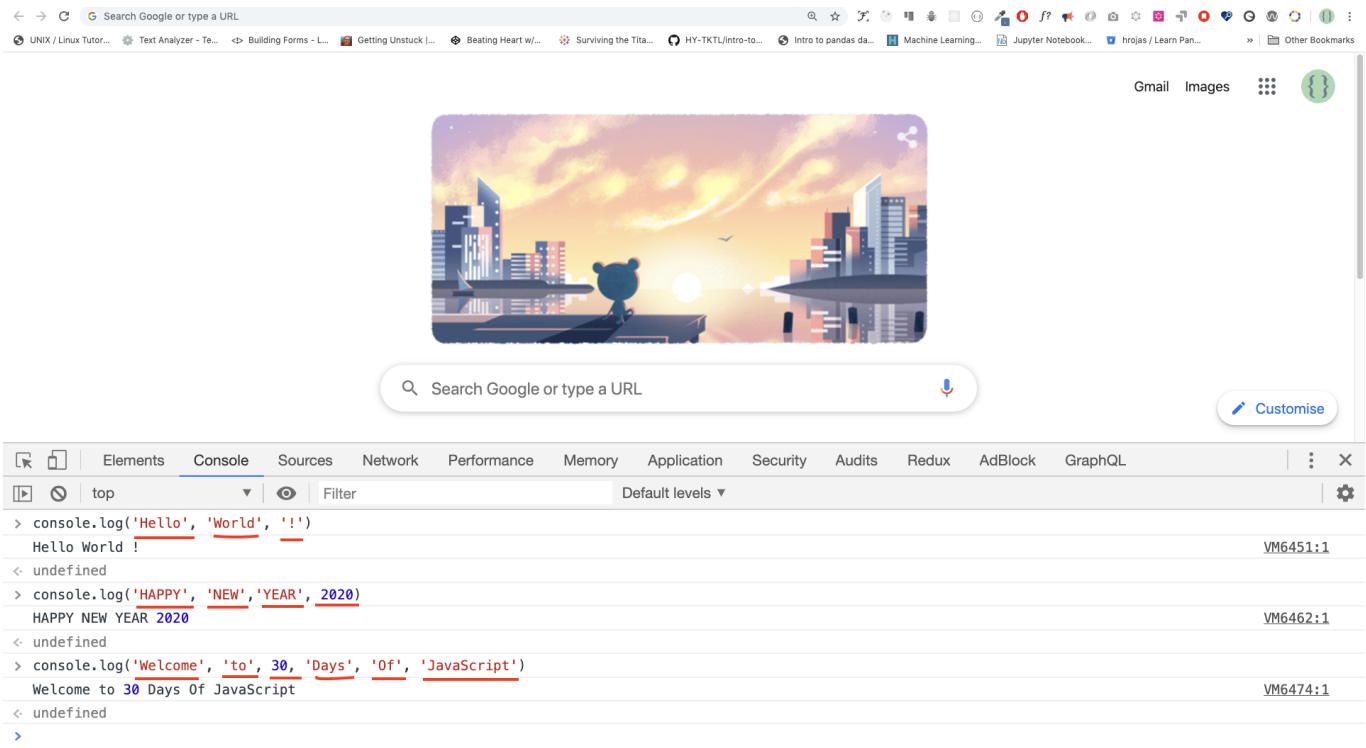
Console.log

To write our first JavaScript code, we used a built-in function **console.log()**. We passed an argument as input data, and the function displays the output. We passed `'Hello, World'` as input data or argument in the console.log() function.

```
console.log('Hello, World!')
```

Console.log with Multiple Arguments

The `console.log()` function can take multiple parameters separated by commas. The syntax looks like as follows:`console.log(param1, param2, param3)`




```
console.log('Hello', 'World', '!')
console.log('HAPPY', 'NEW', 'YEAR', 2020)
console.log('Welcome', 'to', 30, 'Days', 'Of', 'JavaScript')
```

As you can see from the snippet code above, `console.log()` can take multiple arguments.

Congratulations! You wrote your first JavaScript code using `console.log()`.

Comments

We can add comments to our code. Comments are very important to make code more readable and to leave remarks in our code. JavaScript does not execute the comment part of our code. In JavaScript, any text line starting with // in JavaScript is a comment, and anything enclosed like this `//` is also a comment.

Example: Single Line Comment

```
// This is the first comment
// This is the second comment
// I am a single line comment
```

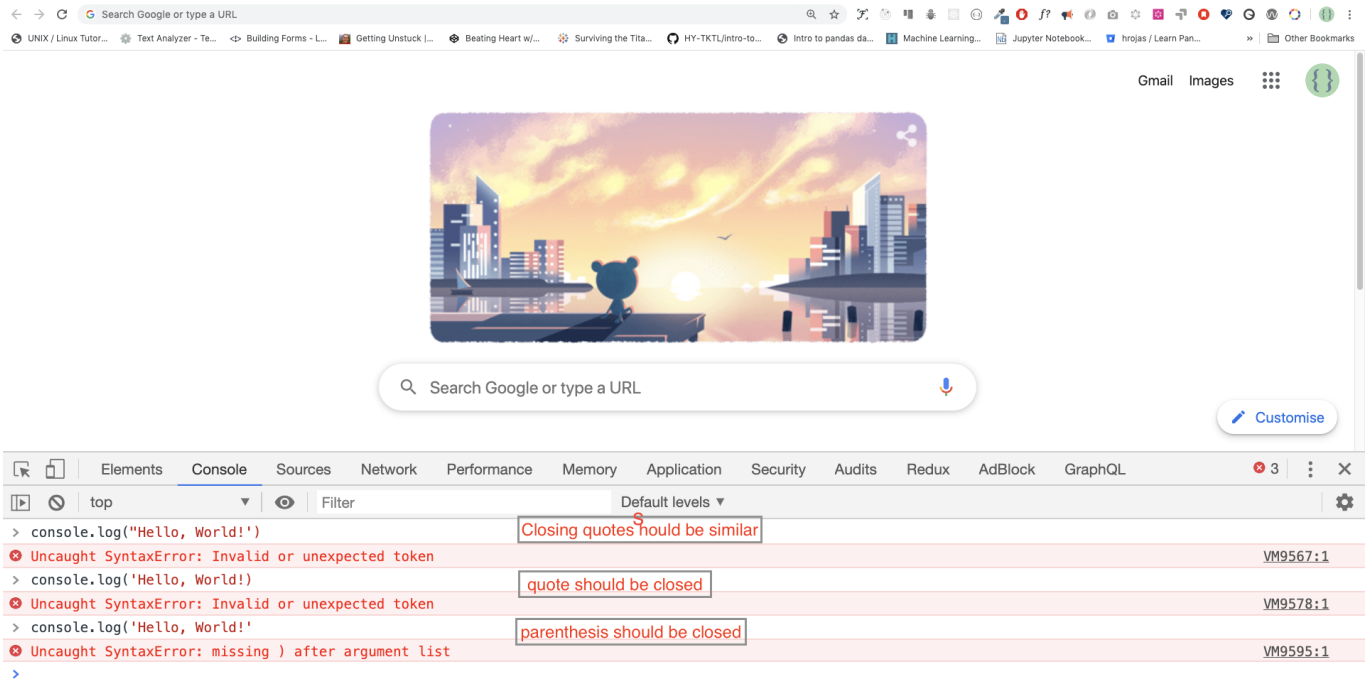
Example: Multiline Comment

```
/*
This is a multiline comment
 Multiline comments can take multiple lines
```

```
JavaScript is the language of the web
*/
```

Syntax

Programming languages are similar to human languages. English or many other language uses words, phrases, sentences, compound sentences and other more to convey a meaningful message. The English meaning of syntax is *the arrangement of words and phrases to create well-formed sentences in a language*. The technical definition of syntax is the structure of statements in a computer language. Programming languages have syntax. JavaScript is a programming language and like other programming languages it has its own syntax. If we do not write a syntax that JavaScript understands, it will raise different types of errors. We will explore different kinds of JavaScript errors later. For now, let us see syntax errors.



I made a deliberate mistake. As a result, the console raises syntax errors. Actually, the syntax is very informative. It informs what type of mistake was made. By reading the error feedback guideline, we can correct the syntax and fix the problem. The process of identifying and removing errors from a program is called debugging. Let us fix the errors:

```
console.log('Hello, World!')
console.log('Hello, World!')
```

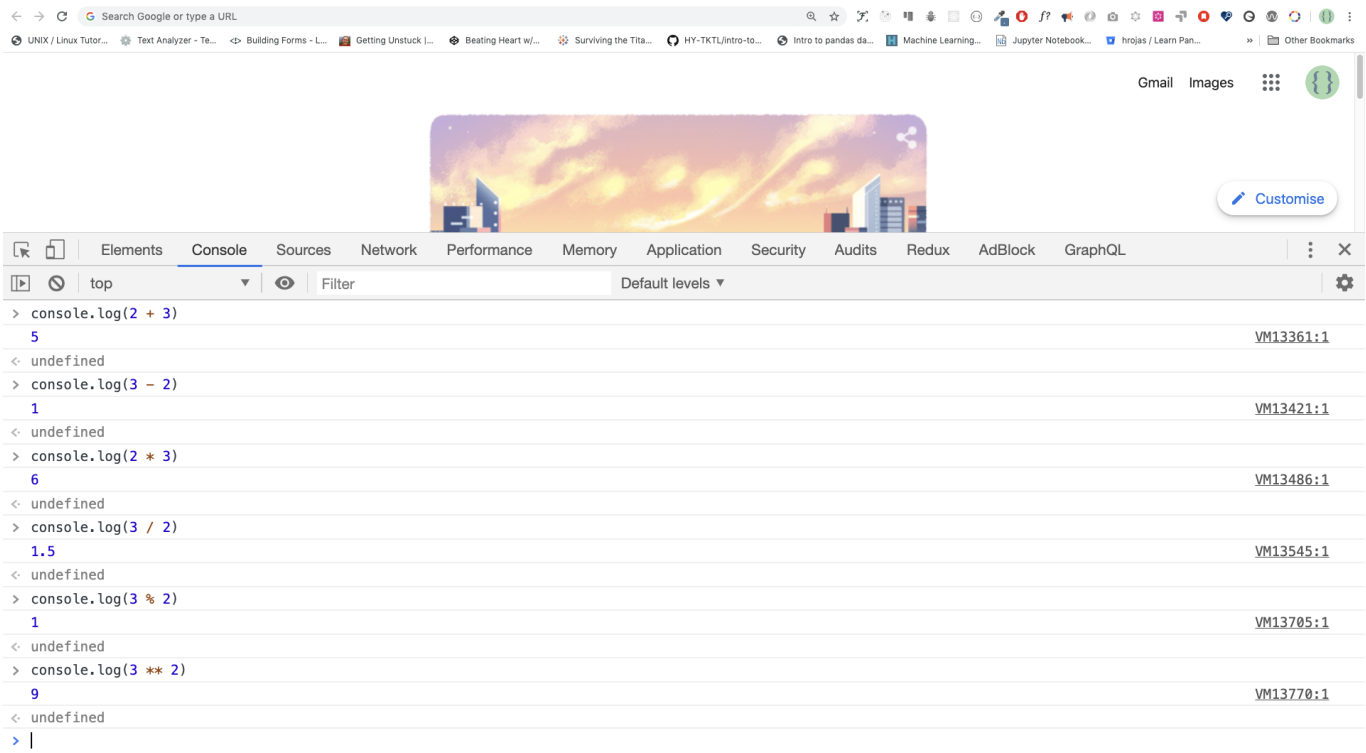
So far, we saw how to display text using the `console.log()`. If we are printing text or string using `console.log()`, the text has to be inside the single quotes, double quotes, or a backtick. **Example:**

```
```

```
console.log('Hello, World!')
console.log("Hello, World!")
console.log(`Hello, World!`)
```

Arithmetics

Now, let us practice more writing JavaScript codes using `console.log()` on Google Chrome console for number data types. In addition to the text, we can also do mathematical calculations using JavaScript. Let us do the following simple calculations. It is possible to write JavaScript code on Google Chrome console can directly without the `console.log()` function. However, it is included in this introduction because most of this challenge would be taking place in a text editor where the usage of the function would be mandatory. You can play around directly with instructions on the console.




```
console.log(2 + 3) // Addition
console.log(3 - 2) // Subtraction
console.log(2 * 3) // Multiplication
console.log(3 / 2) // Division
console.log(3 % 2) // Modulus - finding remainder
console.log(3 ** 2) // Exponentiation 3 ** 2 == 3 * 3
```

Adding JavaScript to a Web Page

JavaScript can be added to a web page in three different ways:

- ***Inline script***
- ***Internal script***

- ***External script***
- ***Multiple External scripts***

The following sections show different ways of adding JavaScript code to your web page.

Inline Script

Create a project folder on your desktop or in any location, name it 30DaysOfJS and create an *index.html* file in the project folder. Then paste the following code and open it in a browser, for example Chrome.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript:Inline Script</title>
  </head>
  <body>
    <button onclick="alert('Welcome to JavaScript!')">Click Me</button>
  </body>
</html>
```

Now, you just wrote your first inline script. We can create a pop up alert message using the *alert()* built-in function.

Internal Script

The internal script can be written in the *head* or the *body*, but it is preferred to put it on the body of the HTML document. First, let us write on the head part of the page.

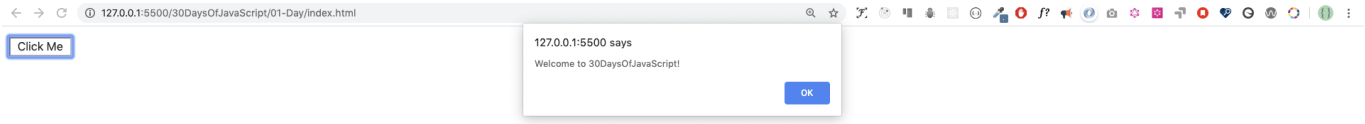
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript:Internal Script</title>
    <script>
      console.log('Welcome to JavaScript')
    </script>
  </head>
  <body></body>
</html>
```

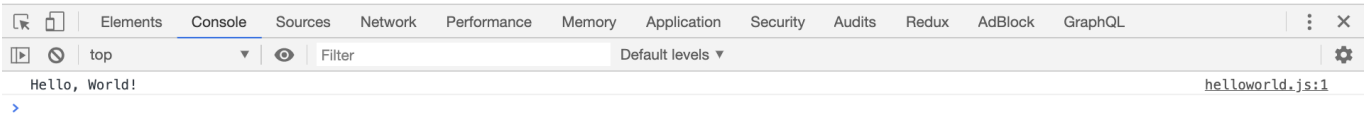
This is how we write an internal script most of the time. Writing the JavaScript code in the body section is the most preferred option. Open the browser console to see the output from the `console.log()`.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript:Internal Script</title>
  </head>
```

```
    <body>
      <button onclick="alert('Welcome to JavaScript!');">Click Me</button>
      <script>
        console.log('Welcome to JavaScript')
      </script>
    </body>
  </html>
```

Open the browser console to see the output from the `console.log()`.





External Script

Similar to the internal script, the external script link can be on the header or body, but it is preferred to put it in the body. First, we should create an external JavaScript file with .js extension. All files ending with .js extension are JavaScript files. Create a file named introduction.js inside your project directory and write the following code and link this .js file at the bottom of the body.

```
console.log('Welcome to JavaScript')
```

External scripts in the *head*:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript:External script</title>
    <script src="introduction.js"></script>
```

```
  </head>
  <body></body>
</html>
```

External scripts in the *body*:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript:External script</title>
  </head>
  <body>
    <!-- JavaScript external link could be in the header or in the body -->
    <!-- Before the closing tag of the body is the recommended place to put the
external JavaScript script -->
    <script src="introduction.js"></script>
  </body>
</html>
```

Open the browser console to see the output of the `console.log()`.

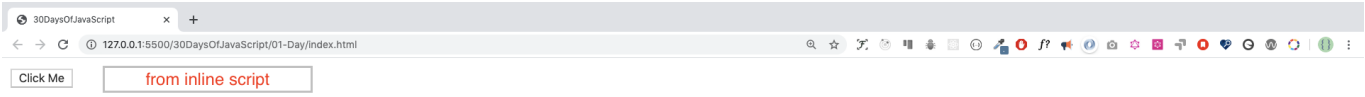
Multiple External Scripts

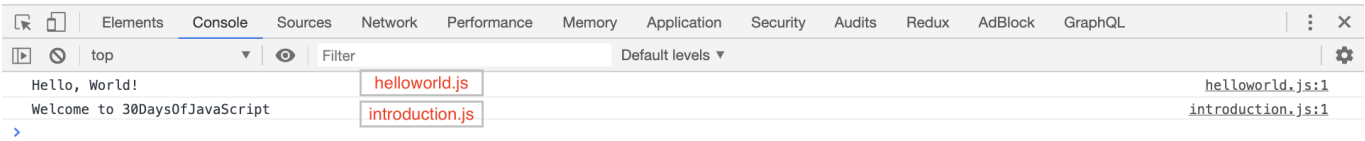
We can also link multiple external JavaScript files to a web page. Create a `helloworld.js` file inside the 30DaysOfJS folder and write the following code.

```
console.log('Hello, World!')
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Multiple External Scripts</title>
  </head>
  <body>
    <script src="./helloworld.js"></script>
    <script src="./introduction.js"></script>
  </body>
</html>
```

Your main.js file should be below all other scripts. It is very important to remember this.





Introduction to Data types

In JavaScript and also other programming languages, there are different types of data types. The following are JavaScript primitive data types: *String, Number, Boolean, undefined, Null*, and *Symbol*.

Numbers

- Integers: Integer (negative, zero and positive) numbers Example: ... -3, -2, -1, 0, 1, 2, 3 ...
- Float-point numbers: Decimal number Example ... -3.5, -2.25, -1.0, 0.0, 1.1, 2.2, 3.5 ...

Strings

A collection of one or more characters between two single quotes, double quotes, or backticks.

Example:

```
'a'
'Computer'
"Computer"
'Surat'
'JavaScript is a beautiful programming language'
'I love teaching'
'I hope you are enjoying the first day'
`We can also create a string using a backtick`
'A string could be just as small as one character or as big as many pages'
'Any data type under a single quote, double quote or backtick is a string'
```

Booleans

A boolean value is either True or False. Any comparisons returns a boolean value, which is either true or false.

A boolean data type is either a true or false value.

Example:

```
true // if the light is on, the value is true
false // if the light is off, the value is false
```

Undefined

In JavaScript, if we don't assign a value to a variable, the value is undefined. In addition to that, if a function is not returning anything, it returns undefined.

```
let firstName
console.log(firstName) // undefined, because it is not assigned to a value yet
```

Null

Null in JavaScript means an empty value.

```
let emptyValue = null
```

Checking Data Types

To check the data type of a certain variable, we use the **typeof** operator. See the following example.

```
console.log(typeof 'Computer') // string
console.log(typeof 5) // number
console.log(typeof true) // boolean
console.log(typeof null) // object type
console.log(typeof undefined) // undefined
```

Comments Again

Remember that commenting in JavaScript is similar to other programming languages. Comments are important in making your code more readable. There are two ways of commenting:

- *Single line commenting*
- *Multiline commenting*

```
// commenting the code itself with a single comment
// let firstName = 'Computer'; single line comment
// let lastName = 'Engineering'; single line comment
```

Multiline commenting:

```
/*
  let location = 'Helsinki';
  let age = 100;
  let isMarried = true;
  This is a Multiple line comment
*/
```

Variables

Variables are *containers* of data. Variables are used to *store* data in a memory location. When a variable is declared, a memory location is reserved. When a variable is assigned to a value (data), the memory space will be filled with that data. To declare a variable, we use *var*, *let*, or *const* keywords.

For a variable that changes at a different time, we use *let*. If the data does not change at all, we use *const*. For example, PI, country name, gravity do not change, and we can use *const*. We will not use var in this challenge and I don't recommend you to use it. It is error prone way of declaring variable it has lots of leak. We will talk more about var, let, and const in detail in other sections (scope). For now, the above explanation is enough.

A valid JavaScript variable name must follow the following rules:

- A JavaScript variable name should not begin with a number.
- A JavaScript variable name does not allow special characters except dollar sign and underscore.
- A JavaScript variable name follows a camelCase convention.
- A JavaScript variable name should not have space between words.

The following are examples of valid JavaScript variables.

```
firstName
lastName
country
city
capitalCity
age
isMarried

first_name
last_name
is_married
capital_city

num1
```

```
num_1
_num_1
$num1
year2020
year_2020
```

The first and second variables on the list follows the camelCase convention of declaring in JavaScript. In this material, we will use camelCase variables(camelWithOneHump). We use CamelCase(CamelWithTwoHump) to declare classes, we will discuss about classes and objects in other section.

Example of invalid variables:

```
  first-name
  1_num
  num_#_1
```

Let us declare variables with different data types. To declare a variable, we need to use *let* or *const* keyword before the variable name. Following the variable name, we write an equal sign (assignment operator), and a value(assigned data).

```
// Syntax
let nameOfVariable = value
```

The nameOfVriable is the name that stores different data of value. See below for detail examples.

Examples of declared variables

```
// Declaring different variables of different data types
let firstName = 'Computer' // first name of a person
let lastName = 'Engineering' // last name of a person
let country = 'Surat' // country
let city = 'Helsinki' // capital city
let age = 100 // age in years
let isMarried = true

console.log(firstName, lastName, country, city, age, isMarried)
```

```
Computer Engineering Surat Helsinki 100 true
```

```
// Declaring variables with number values
let age = 100 // age in years
const gravity = 9.81 // earth gravity  in m/s2
```

```
const boilingPoint = 100 // water boiling point, temperature in °C
const PI = 3.14 // geometrical constant
console.log(gravity, boilingPoint, PI)
```

```
9.81 100 3.14
```

```
// Variables can also be declaring in one line separated by comma, however I
recommend to use a seperate line to make code more readble
let name = 'Computer', job = 'teacher', live = 'Surat'
console.log(name, job, live)
```

```
Computer teacher Surat
```

Data Types

In the previous section, we mentioned a little bit about data types. Data or values have data types. Data types describe the characteristics of data. Data types can be divided into two:

1. Primitive data types
2. Non-primitive data types(Object References)

Primitive Data Types

Primitive data types in JavaScript include:

1. Numbers - Integers, floats
2. Strings - Any data under single quote, double quote or backtick quote
3. Booleans - true or false value
4. Null - empty value or no value
5. Undefined - a declared variable without a value
6. Symbol - A unique value that can be generated by Symbol constructor

Non-primitive data types in JavaScript includes:

1. Objects
2. Arrays

Now, let us see what exactly primitive and non-primitive data types mean. *Primitive* data types are immutable(non-modifiable) data types. Once a primitive data type is created we cannot modify it.

Example:

```
let word = 'JavaScript'
```

If we try to modify the string stored in variable *word*, JavaScript should raise an error. Any data type under a single quote, double quote, or backtick quote is a string data type.

```
word[0] = 'Y'
```

This expression does not change the string stored in the variable *word*. So, we can say that strings are not modifiable or in other words immutable. Primitive data types are compared by its values. Let us compare different data values. See the example below:

```
let numOne = 3
let numTwo = 3

console.log(numOne == numTwo)      // true

let js = 'JavaScript'
let py = 'Python'

console.log(js == py)             //false

let lightOn = true
let lightOff = false

console.log(lightOn == lightOff) // false
```

Non-Primitive Data Types

Non-primitive data types are modifiable or mutable. We can modify the value of non-primitive data types after it gets created. Let us see by creating an array. An array is a list of data values in a square bracket. Arrays can contain the same or different data types. Array values are referenced by their index. In JavaScript array index starts at zero. I.e., the first element of an array is found at index zero, the second element at index one, and the third element at index two, etc.

```
let nums = [1, 2, 3]
nums[0] = 10

console.log(nums)  // [10, 2, 3]
```

As you can see, an array, which is a non-primitive data type is mutable. Non-primitive data types cannot be compared by value. Even if two non-primitive data types have the same properties and values, they are not strictly equal.

```
let nums = [1, 2, 3]
let numbers = [1, 2, 3]
```

```
console.log(nums == numbers)  // false

let userOne = {
name:'Computer',
role:'teaching',
country:'Surat'
}

let userTwo = {
name:'Computer',
role:'teaching',
country:'Surat'
}

console.log(userOne == userTwo) // false
```

Rule of thumb, we do not compare non-primitive data types. Do not compare arrays, functions, or objects. Non-primitive values are referred to as reference types, because they are being compared by reference instead of value. Two objects are only strictly equal if they refer to the same underlying object.

```
let nums = [1, 2, 3]
let numbers = nums

console.log(nums == numbers)  // true

let userOne = {
name:'Computer',
role:'teaching',
country:'Surat'
}

let userTwo = userOne

console.log(userOne == userTwo)  // true
```

If you have a hard time understanding the difference between primitive data types and non-primitive data types, you are not the only one. Calm down and just go to the next section and try to come back after some time. Now let us start the data types by number type.

Numbers

Numbers are integers and decimal values which can do all the arithmetic operations. Let's see some examples of Numbers.

Declaring Number Data Types

```
let age = 35
const gravity = 9.81  // we use const for non-changing values, gravitational
```

```
constant in  m/s2
let mass = 72         // mass in Kilogram
const PI = 3.14       // pi a geometrical constant

// More Examples
const boilingPoint = 100 // temperature in oC, boiling point of water which is a
constant
const bodyTemp = 37      // oC average human body temperature, which is a constant

console.log(age, gravity, mass, PI, boilingPoint, bodyTemp)
```

Math Object

In JavaScript the Math Object provides a lots of methods to work with numbers.

```
const PI = Math.PI

console.log(PI)                              // 3.141592653589793

// Rounding to the closest number
// if above .5 up if less 0.5 down rounding

console.log(Math.round(PI))                  // 3 to round values to the nearest
number

console.log(Math.round(9.81))                // 10

console.log(Math.floor(PI))                  // 3 rounding down

console.log(Math.ceil(PI))                   // 4 rounding up

console.log(Math.min(-5, 3, 20, 4, 5, 10)) // -5, returns the minimum value

console.log(Math.max(-5, 3, 20, 4, 5, 10)) // 20, returns the maximum value

const randNum = Math.random() // creates random number between 0 to 0.999999
console.log(randNum)

// Let us  create random number between 0 to 10

const num = Math.floor(Math.random () * 11) // creates random number between 0 and
10
console.log(num)

//Absolute value
console.log(Math.abs(-10))      // 10

//Square root
console.log(Math.sqrt(100))     // 10

console.log(Math.sqrt(2))       // 1.4142135623730951
```

```
// Power
console.log(Math.pow(3, 2))     // 9

console.log(Math.E)             // 2.718

// Logarithm
// Returns the natural logarithm with base E of x, Math.log(x)
console.log(Math.log(2))        // 0.6931471805599453
console.log(Math.log(10))       // 2.302585092994046

// Returns the natural logarithm of 2 and 10 respectively
console.log(Math.LN2)           // 0.6931471805599453
console.log(Math.LN10)          // 2.302585092994046

// Trigonometry
Math.sin(0)
Math.sin(60)

Math.cos(0)
Math.cos(60)
```

Random Number Generator

The JavaScript Math Object has a random() method number generator which generates number from 0 to 0.999999999...

```
let randomNum = Math.random() // generates 0 to 0.999...
```

Now, let us see how we can use random() method to generate a random number between 0 and 10:

```
let randomNum = Math.random()         // generates 0 to 0.999
let numBtnZeroAndTen = randomNum * 11

console.log(numBtnZeroAndTen)         // this gives: min 0 and max 10.99

let randomNumRoundToFloor = Math.floor(numBtnZeroAndTen)
console.log(randomNumRoundToFloor)    // this gives between 0 and 10
```

Strings

Strings are texts, which are under ***single*** , ***double***, ***back-tick*** quote. To declare a string, we need a variable name, assignment operator, a value under a single quote, double quote, or backtick quote. Let's see some examples of strings:

```

```

```
let space = ' '           // an empty space string
let firstName = 'Computer'
let lastName = 'Engineering'
let country = 'Surat'
let city = 'Helsinki'
let language = 'JavaScript'
let job = 'teacher'
let quote = "The saying,'Seeing is Believing' is not correct in 2020."
let quotWithBackTick = `The saying,'Seeing is Believing' is not correct in 2020.`
```

String Concatenation

Connecting two or more strings together is called concatenation. Using the strings declared in the previous String section:

```
let fullName = firstName + space + lastName; // concatenation, merging two string
together.
console.log(fullName);
```

```
Computer Engineering
```

We can concatenate strings in different ways.

Concatenating Using Addition Operator

Concatenating using the addition operator is an old way. This way of concatenating is tedious and error-prone. It is good to know how to concatenate this way, but I strongly suggest to use the ES6 template strings (explained later on).

```
// Declaring different variables of different data types
let space = ' '
let firstName = 'Computer'
let lastName = 'Engineering'
let country = 'Surat'
let city = 'Helsinki'
let language = 'JavaScript'
let job = 'teacher'
let age = 250


let fullName =firstName + space + lastName
let personInfoOne = fullName + '. I am ' + age + '. I live in ' + country; // ES5
string addition

console.log(personInfoOne)
```

```
Computer Engineering. I am 250. I live in Surat
```

Long Literal Strings

A string could be a single character or paragraph or a page. If the string length is too big it does not fit in one line. We can use the backslash character (\) at the end of each line to indicate that the string will continue on the next line. **Example:**

```
const paragraph = "My name is Computer Engineering. I live in Surat, Helsinki.\
I am a teacher and I love teaching. I teach HTML, CSS, JavaScript, React, Redux, \
Node.js, Python, Data Analysis and D3.js for anyone who is interested to learn. \
In the end of 2019, I was thinking to expand my teaching and to reach \
to global audience and I started a Python challenge from November 20 - December
19.\
It was one of the most rewarding and inspiring experience.\
Now, we are in 2020. I am enjoying preparing the Javascript challenge and \
I hope you are enjoying too."

console.log(paragraph)
```

Escape Sequences in Strings

In JavaScript and other programming languages \ followed by some characters is an escape sequence. Let's see the most common escape characters:

- \n: new line
- \t: Tab, means 8 spaces
- \\: Back slash
- \': Single quote (')
- \": Double quote (")

```
console.log('I hope everyone is enjoying the 30 Days Of JavaScript challenge.\nDo
you ?') // line break
console.log('Days\tTopics\tExercises')
console.log('Day 1\t3\t5')
console.log('Day 2\t3\t5')
console.log('Day 3\t3\t5')
console.log('Day 4\t3\t5')
console.log('This is a backslash  symbol (\\)') // To write a backslash
console.log('In every programming language it starts with \"Hello, World!\"')
console.log("In every programming language it starts with \'Hello, World!\'")
console.log('The saying \'Seeing is Believing\' isn\'t correct in 2020')
```

Output in console:

```
I hope everyone is enjoying the 30 Days Of JavaScript challenge.
Do you ?
Days  Topics  Exercises
Day 1 3 5
Day 2 3 5
Day 3 3 5
Day 4 3 5
This is a backslash  symbol (\)
In every programming language it starts with "Hello, World!"
In every programming language it starts with 'Hello, World!'
The saying 'Seeing is Believing' isn't correct in 2020
```

Template Literals (Template Strings)

To create a template strings, we use two back-ticks. We can inject data as expressions inside a template string. To inject data, we enclose the expression with a curly bracket({}) preceded by a $ sign. See the syntax below.

```
//Syntax
`String literal text`
`String literal text ${expression}`
```

Example: 1

```
console.log(`The sum of 2 and 3 is 5`)             // statically writing the data
let a = 2
let b = 3
console.log(`The sum of ${a} and ${b} is ${a + b}`) // injecting the data
dynamically
```

Example:2

```
let firstName = 'Computer'
let lastName = 'Engineering'
let country = 'Surat'
let city = 'Helsinki'
let language = 'JavaScript'
let job = 'teacher'
let age = 250
let fullName = firstName + ' ' + lastName

let personInfoTwo = `I am ${fullName}. I am ${age}. I live in ${country}.` //ES6 -
String interpolation method
let personInfoThree = `I am ${fullName}. I live in ${city}, ${country}. I am a
${job}. I teach ${language}.`
```

```
console.log(personInfoTwo)
console.log(personInfoThree)
```

```
I am Computer Engineering. I am 250. I live in Surat.
I am Computer Engineering. I live in Helsinki, Surat. I am a teacher. I teach
JavaScript.
```

Using a string template or string interpolation method, we can add expressions, which could be a value, or some operations (comparison, arithmetic operations, ternary operation).

```
let a = 2
let b = 3
console.log(`${a} is greater than ${b}: ${a > b}`)
```

```
2 is greater than 3: false
```

String Methods

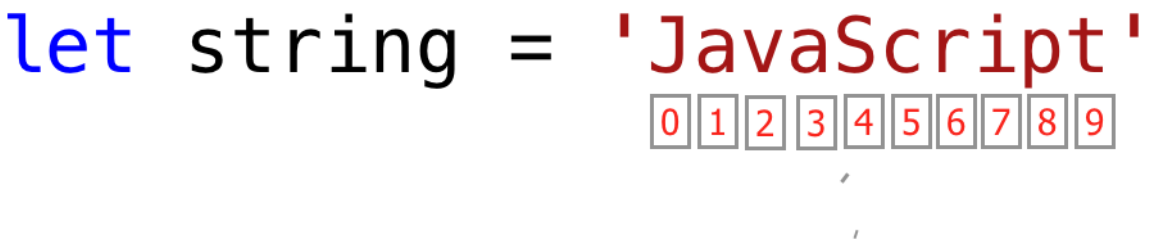
Everything in JavaScript is an object. A string is a primitive data type that means we can not modify it once it is created. The string object has many string methods. There are different string methods that can help us to work with strings.

1. *length*: The string *length* method returns the number of characters in a string included empty space.

Example:

```
let js = 'JavaScript'
console.log(js.length)         // 10
let firstName = 'Computer'
console.log(firstName.length)  // 8
```

2. *Accessing characters in a string*: We can access each character in a string using its index. In programming, counting starts from 0. The first index of the string is zero, and the last index is the length of the string minus one.



Let us access different characters in 'JavaScript' string.

```
let string = 'JavaScript'
let firstLetter = string[0]

console.log(firstLetter)           // J

let secondLetter = string[1]       // a
let thirdLetter = string[2]
let lastLetter = string[9]

console.log(lastLetter)            // t

let lastIndex = string.length - 1

console.log(lastIndex)  // 9
console.log(string[lastIndex])    // t
```

3. *toUpperCase()*: this method changes the string to uppercase letters.

```
let string = 'JavaScript'

console.log(string.toUpperCase())     // JAVASCRIPT

let firstName = 'Computer'

console.log(firstName.toUpperCase())  // COMPUTER

let country = 'Surat'

console.log(country.toUpperCase())    // SURAT
```

4. *toLowerCase()*: this method changes the string to lowercase letters.

```
let string = 'JavasCript'

console.log(string.toLowerCase())     // javascript
```

```
let firstName = 'Computer'

console.log(firstName.toLowerCase())  // computer

let country = 'Surat'

console.log(country.toLowerCase())   // Surat
```

5. *substr()*: It takes two arguments, the starting index and number of characters to slice.

```
let string = 'JavaScript'
console.log(string.substr(4,6))    // Script

let country = 'Surat'
console.log(country.substr(3, 4))   // land
```

6. *substring()*: It takes two arguments, the starting index and the stopping index but it doesn't include the character at the stopping index.

```
let string = 'JavaScript'

console.log(string.substring(0,4))     // Java
console.log(string.substring(4,10))    // Script
console.log(string.substring(4))       // Script

let country = 'Surat'

console.log(country.substring(0, 3))   // Fin
console.log(country.substring(3, 7))   // land
console.log(country.substring(3))      // land
```

7. *split()*: The split method splits a string at a specified place.

```
let string = '30 Days Of JavaScript'

console.log(string.split())     // Changes to an array -> ["30 Days Of
JavaScript"]
console.log(string.split(' '))  // Split to an array at space -> ["30", "Days",
"Of", "JavaScript"]

let firstName = 'Computer'

console.log(firstName.split())    // Change to an array - > ["Computer"]
console.log(firstName.split(''))  // Split to an array at each letter ->  ["A",
"s", "a", "b", "e", "n", "e", "h"]
```

```
let countries = 'Surat, Sweden, Norway, Denmark, and Iceland'

console.log(countries.split(','))  // split to any array at comma -> ["Surat", "
Sweden", " Norway", " Denmark", " and Iceland"]
console.log(countries.split(', ')) //  ["Surat", "Sweden", "Norway", "Denmark",
"and Iceland"]
```

8. *trim()*: Removes trailing space in the beginning or the end of a string.

```
let string = '   30 Days Of JavaScript   '

console.log(string)
console.log(string.trim(' '))

let firstName = ' Computer '

console.log(firstName)
console.log(firstName.trim())  // still removes spaces at the beginning and the
end of the string
```

```
   30 Days Of JavasCript
30 Days Of JavasCript
  Computer
Computer
```

9. *includes()*: It takes a substring argument and it checks if substring argument exists in the string. *includes()* returns a boolean. If a substring exist in a string, it returns true, otherwise it returns false.

```
let string = '30 Days Of JavaScript'

console.log(string.includes('Days'))     // true
console.log(string.includes('days'))     // false - it is case sensitive!
console.log(string.includes('Script'))   // true
console.log(string.includes('script'))   // false
console.log(string.includes('java'))     // false
console.log(string.includes('Java'))     // true

let country = 'Surat'

console.log(country.includes('fin'))     // false
console.log(country.includes('Fin'))     // true
console.log(country.includes('land'))    // true
console.log(country.includes('Land'))    // false
```

10. *replace()*: takes as a parameter the old substring and a new substring.

```
string.replace(oldsubstring, newsubstring)
```

```
let string = '30 Days Of JavaScript'
console.log(string.replace('JavaScript', 'Python')) // 30 Days Of Python

let country = 'Surat'
console.log(country.replace('Fin', 'Noman'))       // Nomanland
```

11. *charAt()*: Takes index and it returns the value at that index

```
string.charAt(index)
```

```
let string = '30 Days Of JavaScript'
console.log(string.charAt(0))        // 3

let lastIndex = string.length - 1
console.log(string.charAt(lastIndex)) // t
```

12. *charCodeAt()*: Takes index and it returns char code (ASCII number) of the value at that index

```
string.charCodeAt(index)
```

```
let string = '30 Days Of JavaScript'
console.log(string.charCodeAt(3))        // D ASCII number is 68

let lastIndex = string.length - 1
console.log(string.charCodeAt(lastIndex)) // t ASCII is 116
```

13. *indexOf()*: Takes a substring and if the substring exists in a string it returns the first position of the substring if does not exist it returns -1

```
string.indexOf(substring)
```

```
let string = '30 Days Of JavaScript'

console.log(string.indexOf('D'))          // 3
```

```
console.log(string.indexOf('Days'))       // 3
console.log(string.indexOf('days'))       // -1
console.log(string.indexOf('a'))          // 4
console.log(string.indexOf('JavaScript')) // 11
console.log(string.indexOf('Script'))     //15
console.log(string.indexOf('script'))     // -1
```

14. *lastIndexOf()*: Takes a substring and if the substring exists in a string it returns the last position of the substring if it does not exist it returns -1

```
//syntax
string.lastIndexOf(substring)
```

```
let string = 'I love JavaScript. If you do not love JavaScript what else can you love.'

console.log(string.lastIndexOf('love'))       // 67
console.log(string.lastIndexOf('you'))        // 63
console.log(string.lastIndexOf('JavaScript')) // 38
```

15. *concat()*: it takes many substrings and joins them.

```
string.concat(substring, substring, substring)
```

```
let string = '30'
console.log(string.concat("Days", "Of", "JavaScript")) // Javascript

let country = 'Fin'
console.log(country.concat("land")) // Surat
```

16. *startsWith*: it takes a substring as an argument and it checks if the string starts with that specified substring. It returns a boolean(true or false).

```
//syntax
string.startsWith(substring)
```

```
let string = 'Love is the best to in this world'

console.log(string.startsWith('Love'))   // true
console.log(string.startsWith('love'))   // false
```

```
console.log(string.startsWith('world'))  // false

let country = 'Surat'

console.log(country.startsWith('Fin'))   // true
console.log(country.startsWith('fin'))   // false
console.log(country.startsWith('land'))  //  false
```

17. *endsWith*: it takes a substring as an argument and it checks if the string ends with that specified substring. It returns a boolean(true or false).

```
string.endsWith(substring)
```

```
let string = 'Love is the most powerful feeling in the world'

console.log(string.endsWith('world'))         // true
console.log(string.endsWith('love'))          // false
console.log(string.endsWith('in the world')) // true

let country = 'Surat'

console.log(country.endsWith('land'))         // true
console.log(country.endsWith('fin'))          // false
console.log(country.endsWith('Fin'))          //  false
```

18. *search*: it takes a substring as an argument and it returns the index of the first match. The search value can be a string or a regular expression pattern.

```
string.search(substring)
```

```
let string = 'I love JavaScript. If you do not love JavaScript what else can you
love.'
console.log(string.search('love'))          // 2
console.log(string.search(/javascript/gi))  // 7
```

19. *match*: it takes a substring or regular expression pattern as an argument and it returns an array if there is match if not it returns null. Let us see how a regular expression pattern looks like. It starts with / sign and ends with / sign.

```
let string = 'love'
let patternOne = /love/     // with out any flag
```

```
let patternTwo = /love/gi   // g-means to search in the whole text, i - case
insensitive
```

Match syntax

```
// syntax
string.match(substring)
```

```
let string = 'I love JavaScript. If you do not love JavaScript what else can you
love.'
console.log(string.match('love'))
```

```
["love", index: 2, input: "I love JavaScript. If you do not love JavaScript what
else can you love.", groups: undefined]
```

```
let pattern = /love/gi
console.log(string.match(pattern))   // ["love", "love", "love"]
```

Let us extract numbers from text using a regular expression. This is not the regular expression section, do not panic! We will cover regular expressions later on.

```
let txt = 'In 2019, I ran 30 Days of Python. Now, in 2020 I am super exited to
start this challenge'
let regEx = /\d+/

// d with escape character means d not a normal d instead acts a digit
// + means one or more digit numbers,
// if there is g after that it means global, search everywhere.

console.log(txt.match(regEx))  // ["2", "0", "1", "9", "3", "0", "2", "0", "2",
"0"]
console.log(txt.match(/\d+/g)) // ["2019", "30", "2020"]
```

20. *repeat()*: it takes a number as argument and it returns the repeated version of the string.

```
string.repeat(n)
```

```
let string = 'love'
console.log(string.repeat(10)) // lovelovelovelovelovelovelovelovelovelove
```

Checking Data Types and Casting

Checking Data Types

To check the data type of a certain variable we use the *typeof* method.

Example:

```
// Different javascript data types
// Let's declare different data types

let firstName = 'Computer'      // string
let lastName = 'Engineering'        // string
let country = 'Surat'         // string
let city = 'Helsinki'           // string
let age = 250                   // number, it is not my real age, do not worry
about it
let job                         // undefined, because a value was not assigned

console.log(typeof 'Computer')  // string
console.log(typeof firstName)   // string
console.log(typeof 10)          // number
console.log(typeof 3.14)        // number
console.log(typeof true)        // boolean
console.log(typeof false)       // boolean
console.log(typeof NaN)         // number
console.log(typeof job)         // undefined
console.log(typeof undefined)   // undefined
console.log(typeof null)        // object
```

Changing Data Type (Casting)

- Casting: Converting one data type to another data type. We use *parseInt()*, *parseFloat()*, *Number()*, + *sign*, *str()* When we do arithmetic operations string numbers should be first converted to integer or float if not it returns an error.

String to Int

We can convert string number to a number. Any number inside a quote is a string number. An example of a string number: '10', '5', etc. We can convert string to number using the following methods:

- parseInt()
- Number()
- Plus sign(+)

```
let num = '10'
let numInt = parseInt(num)
console.log(numInt) // 10
```

```
let num = '10'
let numInt = Number(num)

console.log(numInt) // 10
```

```
let num = '10'
let numInt = +num

console.log(numInt) // 10
```

String to Float

We can convert string float number to a float number. Any float number inside a quote is a string float number. An example of a string float number: '9.81', '3.14', '1.44', etc. We can convert string float to number using the following methods:

- parseFloat()
- Number()
- Plus sign(+)

```
let num = '9.81'
let numFloat = parseFloat(num)

console.log(numFloat) // 9.81
```

```
let num = '9.81'
let numFloat = Number(num)

console.log(numFloat) // 9.81
```

```
let num = '9.81'
let numFloat = +num

console.log(numFloat) // 9.81
```

Float to Int

We can convert float numbers to integers. We use the following method to convert float to int:

- parseInt()

```
let num = 9.81
let numInt = parseInt(num)

console.log(numInt) // 9
```

Data Types : Booleans

Booleans

A boolean data type represents one of the two values:*true* or *false*. Boolean value is either true or false. The use of these data types will be clear when you start the comparison operator. Any comparisons return a boolean value which is either true or false.

Example: Boolean Values

```
let isLightOn = true
let isRaining = false
let isHungry = false
let isMarried = true
let truValue = 4 > 3    // true
let falseValue = 4 < 3  // false
```

We agreed that boolean values are either true or false.

Truthy values

- All numbers(positive and negative) are truthy except zero
- All strings are truthy except an empty string ('')
- The boolean true

Falsy values

- 0
- 0n
- null
- undefined
- NaN
- the boolean false
- '', "", ``, empty string

It is good to remember those truthy values and falsy values. In later section, we will use them with conditions to make decisions.

Undefined

If we declare a variable and if we do not assign a value, the value will be undefined. In addition to this, if a function is not returning the value, it will be undefined.

```
let firstName
console.log(firstName) //not defined, because it is not assigned to a value yet
```

Null

```
let empty = null
console.log(empty) // -> null , means no value
```

Operators

Assignment operators

An equal sign in JavaScript is an assignment operator. It uses to assign a variable.

```
let firstName = 'Computer'
let country = 'Surat'
```

Assignment Operators

Operator	Example	Same As	Description
=	x = y	x = y	y is stored in x
+=	x += y	x = x + y	x + y resut is stored in x
-=	x -= y	x = x - y	x - y resut is stored in x
*=	x *= y	x = x * y	x * y resut is stored in x
/=	x /= y	x = x / y	x / y resut is stored in x
%=	x %= y	x = x % y	x % y resut is stored in x
**=	x **= y	x = x ** y	x ** y resut is stored in x

Arithmetic Operators

Arithmetic operators are mathematical operators.

- Addition(+): a + b
- Subtraction(-): a - b
- Multiplication(*): a * b
- Division(/): a / b

- Modulus(%): a % b
- Exponential(**): a ** b

```
let numOne = 4
let numTwo = 3
let sum = numOne + numTwo
let diff = numOne - numTwo
let mult = numOne * numTwo
let div = numOne / numTwo
let remainder = numOne % numTwo
let powerOf = numOne ** numTwo

console.log(sum, diff, mult, div, remainder, powerOf) // 7,1,12,1.33,1, 64
```

```
const PI = 3.14
let radius = 100          // length in meter

//Let us calculate area of a circle
const areaOfCircle = PI * radius * radius
console.log(areaOfCircle)  //  314 m


const gravity = 9.81      // in m/s2
let mass = 72             // in Kilogram

// Let us calculate weight of an object
const weight = mass * gravity
console.log(weight)        // 706.32 N(Newton)

const boilingPoint = 100  // temperature in oC, boiling point of water
const bodyTemp = 37       // body temperature in oC


// Concatenating string with numbers using string interpolation
/*
 The boiling point of water is 100 oC.
 Human body temperature is 37 oC.
 The gravity of earth is 9.81 m/s2.
 */
console.log(
  `The boiling point of water is ${boilingPoint} oC.\nHuman body temperature is
${bodyTemp} oC.\nThe gravity of earth is ${gravity} m / s2.`
)
```

Comparison Operators

In programming we compare values, we use comparison operators to compare two values. We check if a value is greater or less or equal to other value.

Operator	Name	Example
==	Equal in value only:Equivalent	x == y
===	Equal in value and data type:Exactly equal	x === y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example: Comparison Operators

```
console.log(3 > 2)              // true, because 3 is greater than 2
console.log(3 >= 2)             // true, because 3 is greater than 2
console.log(3 < 2)              // false,  because 3 is greater than 2
console.log(2 < 3)              // true, because 2 is less than 3
console.log(2 <= 3)             // true, because 2 is less than 3
console.log(3 == 2)             // false, because 3 is not equal to 2
console.log(3 != 2)             // true, because 3 is not equal to 2
console.log(3 == '3')           // true, compare only value
console.log(3 === '3')          // false, compare both value and data type
console.log(3 !== '3')          // true, compare both value and data type
console.log(3 != 3)             // false, compare only value
console.log(3 !== 3)            // false, compare both value and data type
console.log(0 == false)         // true, equivalent
console.log(0 === false)        // false, not exactly the same
console.log(0 == '')            // true, equivalent
console.log(0 == ' ')           // true, equivalent
console.log(0 === '')           // false, not exactly the same
console.log(1 == true)          // true, equivalent
console.log(1 === true)         // false, not exactly the same
console.log(undefined == null)  // true
console.log(undefined === null) // false
console.log(NaN == NaN)         // false, not equal
console.log(NaN === NaN)        // false
console.log(typeof NaN)         // number

console.log('mango'.length == 'avocado'.length)  // false
console.log('mango'.length != 'avocado'.length)  // true
console.log('mango'.length < 'avocado'.length)   // true
console.log('milk'.length == 'meat'.length)      // true
console.log('milk'.length != 'meat'.length)      // false
console.log('tomato'.length == 'potato'.length)  // true
console.log('python'.length > 'dragon'.length)   // false
```

Try to understand the above comparisons with some logic. Remembering without any logic might be difficult. JavaScript is somehow a wired kind of programming language. JavaScript code run and give you a result but

unless you are good at it may not be the desired result.

As rule of thumb, if a value is not true with == it will not be equal with ===. Using === is safer than using ==. The following link has an exhaustive list of comparison of data types.

Logical Operators

The following symbols are the common logical operators: &&(ampersand) , ||(pipe) and !(negation). The && operator gets true only if the two operands are true. The || operator gets true either of the operand is true. The ! operator negates true to false and false to true.

```
// && ampersand operator example

const check = 4 > 3 && 10 > 5         // true && true -> true
const check = 4 > 3 && 10 < 5         // true && false -> false
const check = 4 < 3 && 10 < 5         // false && false -> false

// || pipe or operator, example

const check = 4 > 3 || 10 > 5         // true  || true -> true
const check = 4 > 3 || 10 < 5         // true  || false -> true
const check = 4 < 3 || 10 < 5         // false || false -> false

//! Negation examples

let check = 4 > 3                     // true
let check = !(4 > 3)                  //  false
let isLightOn = true
let isLightOff = !isLightOn           // false
let isMarried = !false                // true
```

Increment Operator

In JavaScript we use the increment operator to increase a value stored in a variable. The increment could be pre or post increment. Let us see each of them:

1. Pre-increment

```
let count = 0
console.log(++count)        // 1
console.log(count)          // 1
```

1. Post-increment

```
let count = 0
console.log(count++)        // 0
console.log(count)          // 1
```

We use most of the time post-increment. At least you should remember how to use post-increment operator.

Decrement Operator

In JavaScript we use the decrement operator to decrease a value stored in a variable. The decrement could be pre or post decrement. Let us see each of them:

1. Pre-decrement

```
let count = 0
console.log(--count) // -1
console.log(count)  // -1
```

2. Post-decrement

```
let count = 0
console.log(count--) // 0
console.log(count)   // -1
```

Ternary Operators

Ternary operator allows to write a condition. Another way to write conditionals is using ternary operators. Look at the following examples:

```
let isRaining = true
isRaining
  ? console.log('You need a rain coat.')
  : console.log('No need for a rain coat.')
isRaining = false

isRaining
  ? console.log('You need a rain coat.')
  : console.log('No need for a rain coat.')
```

```
You need a rain coat.
No need for a rain coat.
```

```
let number = 5
number > 0
  ? console.log(`${number} is a positive number`)
  : console.log(`${number} is a negative number`)
number = -5
```

```
number > 0
  ? console.log(`${number} is a positive number`)
  : console.log(`${number} is a negative number`)
```

```
5 is a positive number
-5 is a negative number
```

Window Methods

Window alert() method

As you have seen at very beginning alert() method displays an alert box with a specified message and an OK button. It is a builtin method and it takes on argument.

```
alert(message)
```

```
alert('Welcome to Javascript')
```

Do not use too much alert because it is destructing and annoying, use it just to test.

Window prompt() method

The window prompt methods display a prompt box with an input on your browser to take input values and the input data can be stored in a variable. The prompt() method takes two arguments. The second argument is optional.

```
prompt('required text', 'optional text')
```

```
let number = prompt('Enter number', 'number goes here')
console.log(number)
```

Window confirm() method

The confirm() method displays a dialog box with a specified message, along with an OK and a Cancel button. A confirm box is often used to ask permission from a user to execute something. Window confirm() takes a string as an argument. Clicking the OK yields true value, whereas clicking the Cancel button yields false value.

```

```

```
const agree = confirm('Are you sure you like to delete? ')
console.log(agree) // result will be true or false based on what you click on the
dialog box
```

These are not all the window methods we will have a separate section to go deep into window methods.

Conditionals

Conditional statements are used for make decisions based on different conditions. By default , statements in JavaScript script executed sequentially from top to bottom. If the processing logic require so, the sequential flow of execution can be altered in two ways:

- Conditional execution: a block of one or more statements will be executed if a certain expression is true
- Repetitive execution: a block of one or more statements will be repetitively executed as long as a certain expression is true. In this section, we will cover *if*, *else* , *else if* statements. The comparison and logical operators we learned in the previous sections will be useful in here.

Conditions can be implementing using the following ways:

- if
- if else
- if else if else
- switch
- ternary operator

If

In JavaScript and other programming languages the key word *if* is to used check if a condition is true and to execute the block code. To create an if condition, we need *if* keyword, condition inside a parenthesis and block of code inside a curly bracket({}).

```
// syntax
if (condition) {
  //this part of code runs for truthy condition
}
```

Example:

```
let num = 3
if (num > 0) {
  console.log(`${num} is a positive number`)
}
//  3 is a positive number
```

As you can see in the condition example above, 3 is greater than 0, so it is a positive number. The condition was true and the block of code was executed. However, if the condition is false, we won't see any results.

```
let isRaining = true
if (isRaining) {
  console.log('Remember to take your rain coat.')
}
```

The same goes for the second condition, if isRaining is false the if block will not be executed and we do not see any output. In order to see the result of a falsy condition, we should have another block, which is going to be *else*.

If Else

If condition is true the first block will be executed, if not the else condition will be executed.

```
// syntax
if (condition) {
  // this part of code runs for truthy condition
} else {
  // this part of code runs for false condition
}
```

```
let num = 3
if (num > 0) {
  console.log(`${num} is a positive number`)
} else {
  console.log(`${num} is a negative number`)
}
//  3 is a positive number

num = -3
if (num > 0) {
  console.log(`${num} is a positive number`)
} else {
  console.log(`${num} is a negative number`)
}
//  -3 is a negative number
```

```
let isRaining = true
if (isRaining) {
  console.log('You need a rain coat.')
} else {
  console.log('No need for a rain coat.')
}
// You need a rain coat.

isRaining = false
```

```
if (isRaining) {
  console.log('You need a rain coat.')
} else {
  console.log('No need for a rain coat.')
}
// No need for a rain coat.
```

The last condition is false, therefore the else block was executed. What if we have more than two conditions? In that case, we would use *else if* conditions.

If Else if Else

On our daily life, we make decisions on daily basis. We make decisions not by checking one or two conditions instead we make decisions based on multiple conditions. As similar to our daily life, programming is also full of conditions. We use *else if* when we have multiple conditions.

```
// syntax
if (condition) {
     // code
} else if (condition) {
   // code
} else {
    //  code

}
```

Example:

```
let a = 0
if (a > 0) {
  console.log(`${a} is a positive number`)
} else if (a < 0) {
  console.log(`${a} is a negative number`)
} else if (a == 0) {
  console.log(`${a} is zero`)
} else {
  console.log(`${a} is not a number`)
}
```

```
// if else if else
let weather = 'sunny'
if (weather === 'rainy') {
  console.log('You need a rain coat.')
} else if (weather === 'cloudy') {
  console.log('It might be cold, you need a jacket.')
} else if (weather === 'sunny') {
  console.log('Go out freely.')
```

```
} else {
  console.log('No need for rain coat.')
}
```

Switch

Switch is an alternative for **if else if else else**. The switch statement starts with a *switch* keyword followed by a parenthesis and code block. Inside the code block we will have different cases. Case block runs if the value in the switch statement parenthesis matches with the case value. The break statement is to terminate execution so the code execution does not go down after the condition is satisfied. The default block runs if all the cases don't satisfy the condition.

```
switch(caseValue){
  case 1:
    // code
    break
  case 2:
   // code
   break
  case 3:
   // code
   break
  default:
   // code
}
```

```
let weather = 'cloudy'
switch (weather) {
  case 'rainy':
    console.log('You need a rain coat.')
    break
  case 'cloudy':
    console.log('It might be cold, you need a jacket.')
    break
  case 'sunny':
    console.log('Go out freely.')
    break
  default:
    console.log(' No need for rain coat.')
}

// Switch More Examples
let dayUserInput = prompt('What day is today ?')
let day = dayUserInput.toLowerCase()

switch (day) {
  case 'monday':
    console.log('Today is Monday')
```

```
    break
  case 'tuesday':
    console.log('Today is Tuesday')
    break
  case 'wednesday':
    console.log('Today is Wednesday')
    break
  case 'thursday':
    console.log('Today is Thursday')
    break
  case 'friday':
    console.log('Today is Friday')
    break
  case 'saturday':
    console.log('Today is Saturday')
    break
  case 'sunday':
    console.log('Today is Sunday')
    break
  default:
    console.log('It is not a week day.')
}
```

// Examples to use conditions in the cases

```
let num = prompt('Enter number');
switch (true) {
  case num > 0:
    console.log('Number is positive');
    break;
  case num == 0:
    console.log('Numbers is zero');
    break;
  case num < 0:
    console.log('Number is negative');
    break;
  default:
    console.log('Entered value was not a number');
}
```

Ternary Operators

Another way to write conditionals is using ternary operators. We have covered this in other sections, but we should also mention it here.

```
let isRaining = true
isRaining
```

```
    ? console.log('You need a rain coat.')
    : console.log('No need for a rain coat.')
```

Arrays

In contrast to variables, an array can store *multiple values*. Each value in an array has an *index*, and each index has *a reference in a memory address*. Each value can be accessed by using their *indexes*. The index of an array starts from *zero*, and the index of the last element is less by one from the length of the array.

An array is a collection of different data types which are ordered and changeable(modifiable). An array allows storing duplicate elements and different data types. An array can be empty, or it may have different data type values.

How to create an empty array

In JavaScript, we can create an array in different ways. Let us see different ways to create an array. It is very common to use *const* instead of *let* to declare an array variable. If you ar using const it means you do not use that variable name again.

- Using Array constructor

```
// syntax
const arr = Array()
// or
// let arr = new Array()
console.log(arr) // []
```

- Using square brackets([])

```
// syntax
// This the most recommended way to create an empty list
const arr = []
console.log(arr)
```

How to create an array with values

Array with initial values. We use *length* property to find the length of an array.

```
const numbers = [0, 3.14, 9.81, 37, 98.6, 100] // array of numbers
const fruits = ['banana', 'orange', 'mango', 'lemon'] // array of strings, fruits
const vegetables = ['Tomato', 'Potato', 'Cabbage', 'Onion', 'Carrot'] // array of
strings, vegetables
const animalProducts = ['milk', 'meat', 'butter', 'yoghurt'] // array of strings,
products
const webTechs = ['HTML', 'CSS', 'JS', 'React', 'Redux', 'Node', 'MongDB'] //
array of web technologies
```

```
const countries = ['Surat', 'Denmark', 'Sweden', 'Norway', 'Iceland'] // array of strings, countries

// Print the array and its length

console.log('Numbers:', numbers)
console.log('Number of numbers:', numbers.length)

console.log('Fruits:', fruits)
console.log('Number of fruits:', fruits.length)

console.log('Vegetables:', vegetables)
console.log('Number of vegetables:', vegetables.length)

console.log('Animal products:', animalProducts)
console.log('Number of animal products:', animalProducts.length)

console.log('Web technologies:', webTechs)
console.log('Number of web technologies:', webTechs.length)

console.log('Countries:', countries)
console.log('Number of countries:', countries.length)
```

```
Numbers: [0, 3.14, 9.81, 37, 98.6, 100]
Number of numbers: 6
Fruits: ['banana', 'orange', 'mango', 'lemon']
Number of fruits: 4
Vegetables: ['Tomato', 'Potato', 'Cabbage', 'Onion', 'Carrot']
Number of vegetables: 5
Animal products: ['milk', 'meat', 'butter', 'yoghurt']
Number of animal products: 4
Web technologies: ['HTML', 'CSS', 'JS', 'React', 'Redux', 'Node', 'MongDB']
Number of web technologies: 7
Countries: ['Surat', 'Estonia', 'Denmark', 'Sweden', 'Norway']
Number of countries: 5
```

- Array can have items of different data types

```
const arr = [
    'Computer',
    250,
    true,
    { country: 'Surat', city: 'Helsinki' },
    { skills: ['HTML', 'CSS', 'JS', 'React', 'Python'] }
] // arr containing different data types
console.log(arr)
```

Loops

Most of the activities we do in life are full of repetitions. Imagine if I ask you to print out from 0 to 100 using console.log(). To implement this simple task it may take you 2 to 5 minutes, such kind of tedious and repetitive task can be carried out using loop.

In programming languages to carry out repetitive task we use different kinds of loops. The following examples are the commonly used loops in JavaScript and other programming languages.

for Loop

```
// For loop structure
for(initialization, condition, increment/decrement){
  // code goes here
}
```

```
for(let i = 0; i <= 5; i++){
  console.log(i)
}

// 0 1 2 3 4 5
```

```
for(let i = 5; i >= 0; i--){
  console.log(i)
}

// 5 4 3 2 1 0
```

```
for(let i = 0; i <= 5; i++){
  console.log(`${i} * ${i} = ${i * i}`)
}
```

```
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
5 * 5 = 25
```

```
const countries = ['Surat', 'Sweden', 'Denmark', 'Norway', 'Iceland']
const newArr = []
for(let i = 0; i < countries.length; i++){
  newArr.push(countries[i].toUpperCase())
```

```
}

// ["SURAT", "SWEDEN", "DENMARK", "NORWAY", "ICELAND"]
```

Adding all elements in the array

```
const numbers = [1, 2, 3, 4, 5]
let sum = 0
for(let i = 0; i < numbers.length; i++){
  sum  = sum + numbers[i]  // can be shorten, sum += numbers[i]

}

console.log(sum)  // 15
```

Creating a new array based on the existing array

```
const numbers = [1, 2, 3, 4, 5]
const newArr = []
let sum = 0
for(let i = 0; i < numbers.length; i++){
  newArr.push( numbers[i] ** 2)

}

console.log(newArr)  // [1, 4, 9, 16, 25]
```

```
const countries = ['Surat', 'Sweden', 'Norway', 'Denmark', 'Iceland']
const newArr = []
for(let i = 0; i < countries.length; i++){
  newArr.push(countries[i].toUpperCase())
}

console.log(newArr)  // ["SURAT", "SWEDEN", "NORWAY", "DENMARK", "ICELAND"]
```

while loop

```
let i = 0
while (i <= 5) {
  console.log(i)
  i++
}

// 0 1 2 3 4 5
```

do while loop

```
let i = 0
do {
  console.log(i)
  i++
} while (i <= 5)

// 0 1 2 3 4 5
```

for of loop

We use for of loop for arrays. It is very hand way to iterate through an array if we are not interested in the index of each element in the array.

```
for (const element of arr) {
  // code goes here
}
```

```
const numbers = [1, 2, 3, 4, 5]

for (const num of numbers) {
  console.log(num)
}

// 1 2 3 4 5

for (const num of numbers) {
  console.log(num * num)
}

// 1 4 9 16 25

// adding all the numbers in the array
let sum = 0
for (const num of numbers) {
  sum = sum + num
    // can be also shorten like this, sum += num
  // after this we will use the shorter synthax(+=, -=, *=, /= etc)
}
console.log(sum) // 15

const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
```

```
  'React',
  'Redux',
  'Node',
  'MongoDB'
]

for (const tech of webTechs) {
  console.log(tech.toUpperCase())
}

// HTML CSS JAVASCRIPT REACT NODE MONGODB

for (const tech of webTechs) {
  console.log(tech[0]) // get only the first letter of each element,  H C J R N M
}
```

```
const countries = ['Surat', 'Sweden', 'Norway', 'Denmark', 'Iceland']
const newArr = []
for(const country of countries){
  newArr.push(country.toUpperCase())
}

console.log(newArr)  // ["SURAT", "SWEDEN", "NORWAY", "DENMARK", "ICELAND"]
```

break

Break is used to interrupt a loop.

```
for(let i = 0; i <= 5; i++){
  if(i == 3){
    break
  }
  console.log(i)
}

// 0 1 2
```

The above code stops if 3 found in the iteration process.

continue

We use the keyword *continue* to skip a certain iterations.

```
for(let i = 0; i <= 5; i++){
  if(i == 3){
    continue
```

```
    }
    console.log(i)
}

// 0 1 2 4 5
```

Functions

So far we have seen many builtin JavaScript functions. In this section, we will focus on custom functions. What is a function? Before we start making functions, lets understand what function is and why we need function?

A function is a reusable block of code or programming statements designed to perform a certain task. A function is declared by a function key word followed by a name, followed by parentheses (). A parentheses can take a parameter. If a function take a parameter it will be called with argument. A function can also take a default parameter. To store a data to a function, a function has to return certain data types. To get the value we call or invoke a function. Function makes code:

- clean and easy to read
- reusable
- easy to test

A function can be declared or created in couple of ways:

- *Declaration function*
- *Expression function*
- *Anonymous function*
- *Arrow function*

We 'll be majorly focusing on ***Declaration Function*** withing this Document.

Function Declaration

Let us see how to declare a function and how to call a function.

```
//declaring a function without a parameter
function functionName() {
  // code goes here
}
functionName() // calling function by its name and with parentheses
```

Function without a parameter and return

Function can be declared without a parameter.

Example:

```
// function without parameter,  a function which make a number square
function square() {
```

```
  let num = 2
  let sq = num * num
  console.log(sq)
}

square() // 4

// function without parameter
function addTwoNumbers() {
  let numOne = 10
  let numTwo = 20
  let sum = numOne + numTwo

  console.log(sum)
}

addTwoNumbers() // a function has to be called by its name to be executed
```

```
  function printFullName (){
      let firstName = 'Computer'
      let lastName = 'Engineering'
      let space = ' '
      let fullName = firstName + space + lastName
      console.log(fullName)
}

printFullName() // calling a function
```

Function returning value

Function can also return values, if a function does not return values the value of the function is undefined. Let us write the above functions with return. From now on, we return value to a function instead of printing it.

```
function printFullName (){
      let firstName = 'Computer'
      let lastName = 'Engineering'
      let space = ' '
      let fullName = firstName + space + lastName
      return fullName
}
console.log(printFullName())
```

```
  function addTwoNumbers() {
      let numOne = 2
      let numTwo = 3
      let total = numOne + numTwo
```

```
        return total

    }

console.log(addTwoNumbers())
```

Function with a parameter

In a function we can pass different data types(number, string, boolean, object, function) as a parameter.

```
// function with one parameter
function functionName(parm1) {
  //code goes her
}
functionName(parm1) // during calling or invoking one argument needed

function areaOfCircle(r) {
  let area = Math.PI * r * r
  return area
}

console.log(areaOfCircle(10)) // should be called with one argument

function square(number) {
  return number * number
}

console.log(square(10))
```

Function with two parameters

```
// function with two parameters
function functionName(parm1, parm2) {
  //code goes her
}
functionName(parm1, parm2) // during calling or invoking two arguments needed
// Function without parameter doesn't take input, so lets make a function with
parameters
function sumTwoNumbers(numOne, numTwo) {
  let sum = numOne + numTwo
  console.log(sum)
}
sumTwoNumbers(10, 20) // calling functions
// If a function doesn't return it doesn't store data, so it should return

function sumTwoNumbers(numOne, numTwo) {
  let sum = numOne + numTwo
  return sum
}
```

```
console.log(sumTwoNumbers(10, 20))
function printFullName(firstName, lastName) {
  return `${firstName} ${lastName}`
}
console.log(printFullName('Computer', 'Engineering'))
```

Function with many parameters

```
// function with multiple parameters
function functionName(parm1, parm2, parm3,...){
  //code goes here
}
functionName(parm1,parm2,parm3,...) // during calling or invoking three arguments
needed


// this function takes array as a parameter and sum up the numbers in the array
function sumArrayValues(arr) {
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    sum = sum + arr[i];
  }
  return sum;
}
const numbers = [1, 2, 3, 4, 5];
    //calling a function
console.log(sumArrayValues(numbers));


    const areaOfCircle = (radius) => {
      let area = Math.PI * radius * radius;
      return area;
    }
console.log(areaOfCircle(10))
```