Beginner to Job-Ready 30 Days Plan

Master JavaScript, Build Projects & Boost Your Career!

30 DAYS OF JAVASCRIPT

30 DAYS OF JAVASCRIPT

Beginner to Advance Guide

TABLE OF CONTENT

DAY-1 INTRODUCTION

Introduction

Congratulations on taking the first step toward mastering JavaScript! This book is designed to help you learn JavaScript from the ground up, whether you're a beginner or an experienced programmer looking to deepen your understanding.

JavaScript is the language of the web, powering interactive websites, mobile applications, desktop software, games, and even server-side development. In recent years, it has become one of the most widely used programming languages, consistently ranking as a top choice among developers worldwide.

This book follows a structured, step-by-step approach to learning JavaScript. The content is presented in a clear and engaging manner, making complex concepts easier to grasp. Along the way, you'll gain practical coding experience and develop problem-solving skills that are essential for any programmer.

By the end of this book, you'll have a solid foundation in JavaScript, equipping you with the knowledge to build interactive web applications and explore advanced topics like backend development, machine learning, and AI.

Get ready to dive in, write code, and enjoy the journey of learning JavaScript!

Requirements

No prior knowledge of programming is required to follow this challenge. You need only:

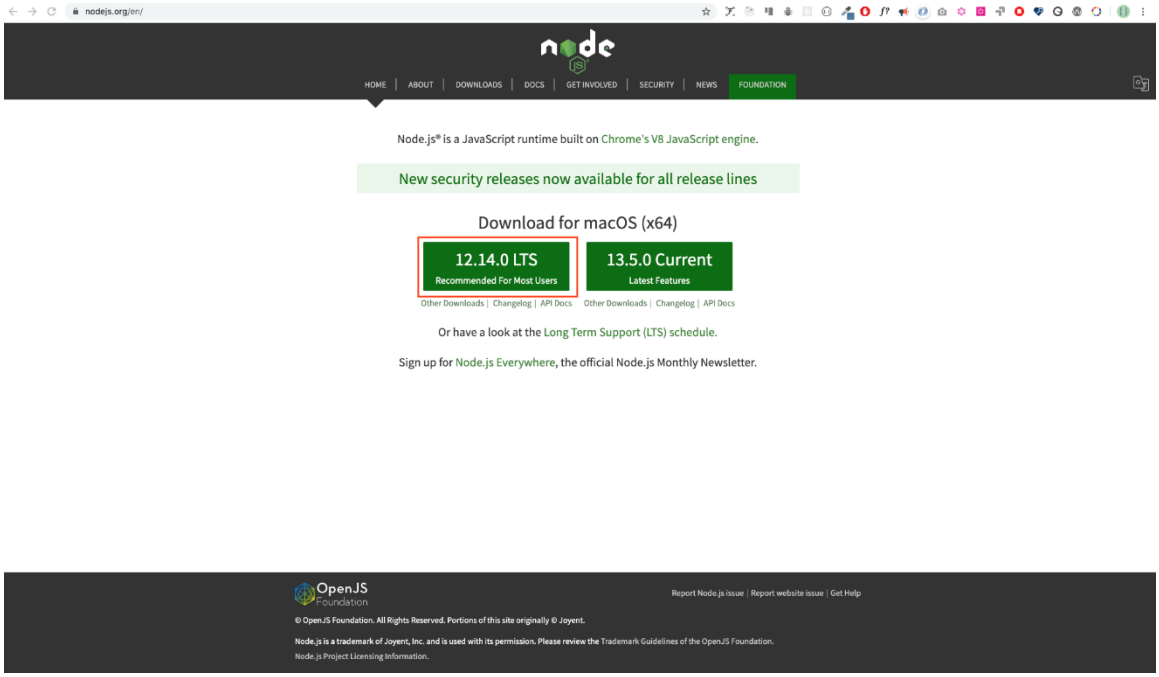
1. Motivation
2. A computer
3. Internet
4. A browser
5. A code editor

Setup

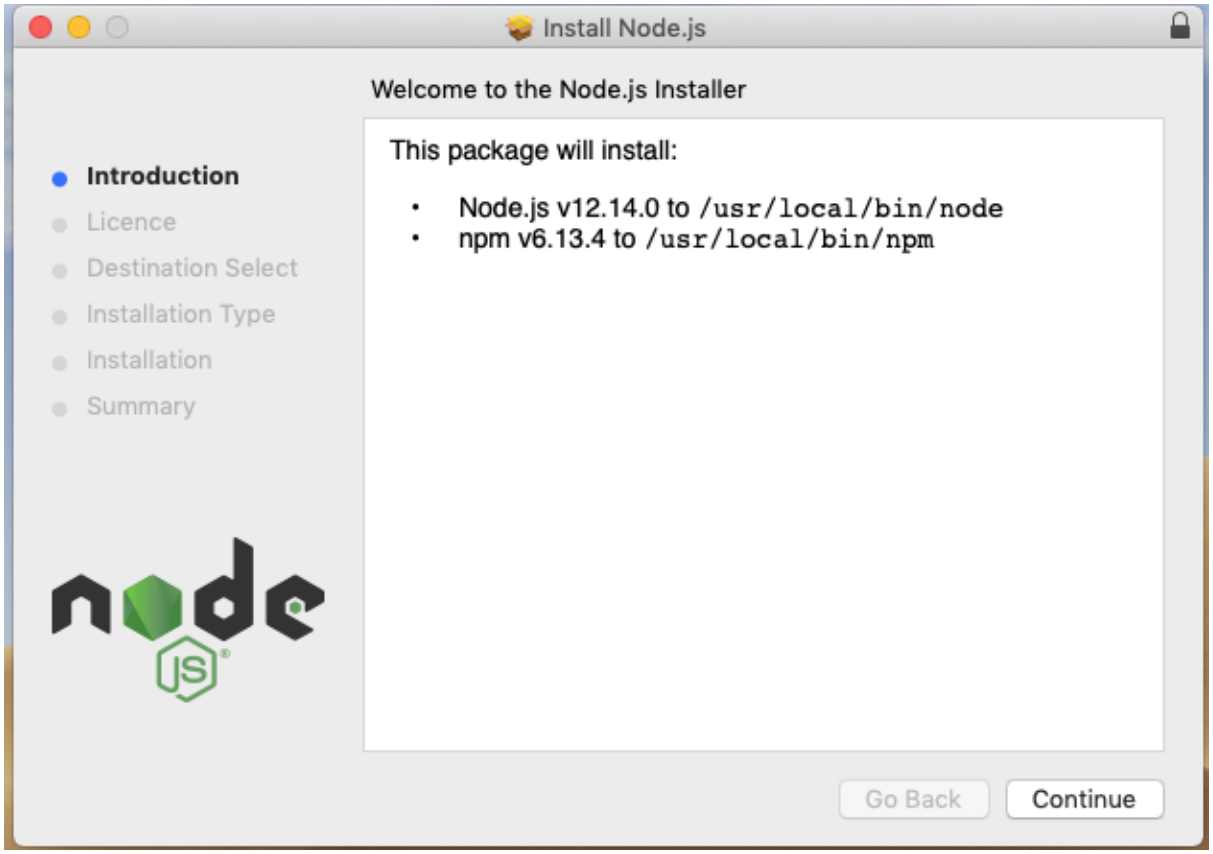
I believe you have the motivation and a strong desire to be a developer, a computer and Internet. If you have those, then you have everything to get started.

Install Node.js

You may not need Node.js right now but you may need it for later. Install node.js.



After downloading double click and install



We can check if node is installed on our local machine by opening our device terminal or command prompt.

```
asabeneh $ node -v

v12.14.0
```

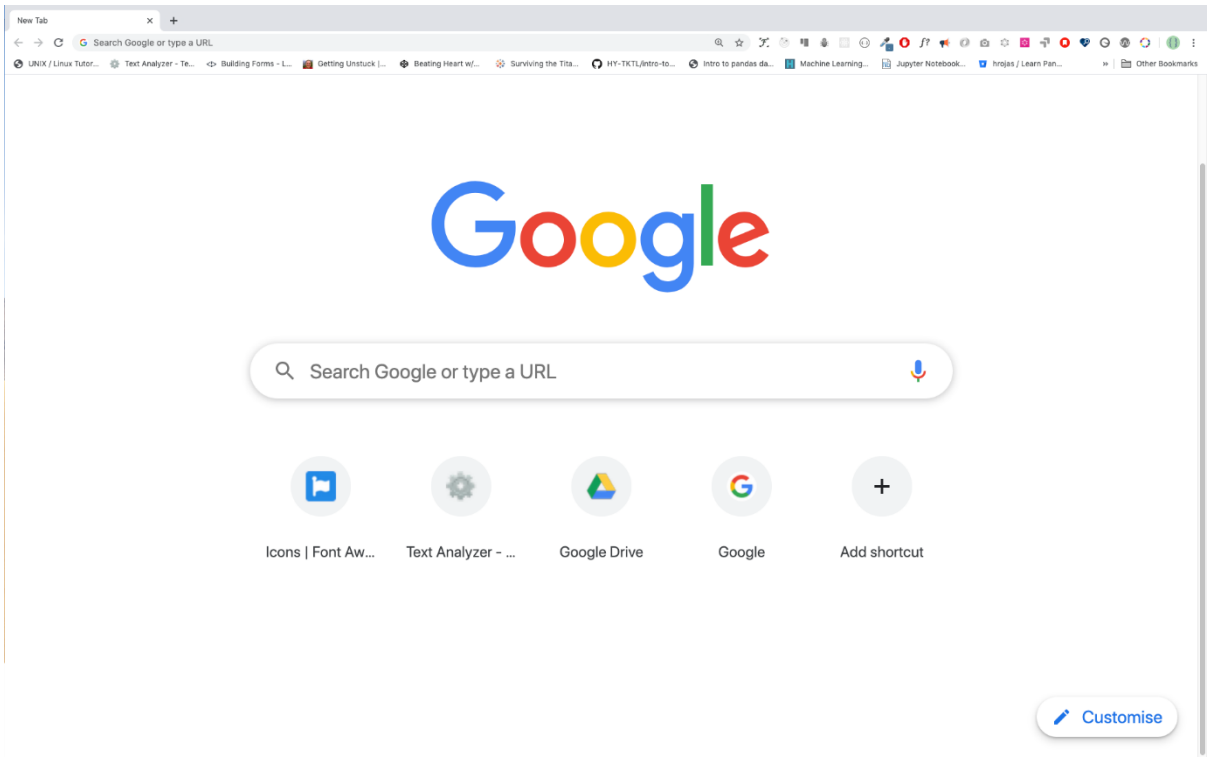
When making this tutorial I was using Node version 12.14.0, but now the recommended version of Node.js for download is v14.17.6, by the time you use this material you may have a higher Node.js version.

Browser

There are many browsers out there. However, I strongly recommend Google Chrome.

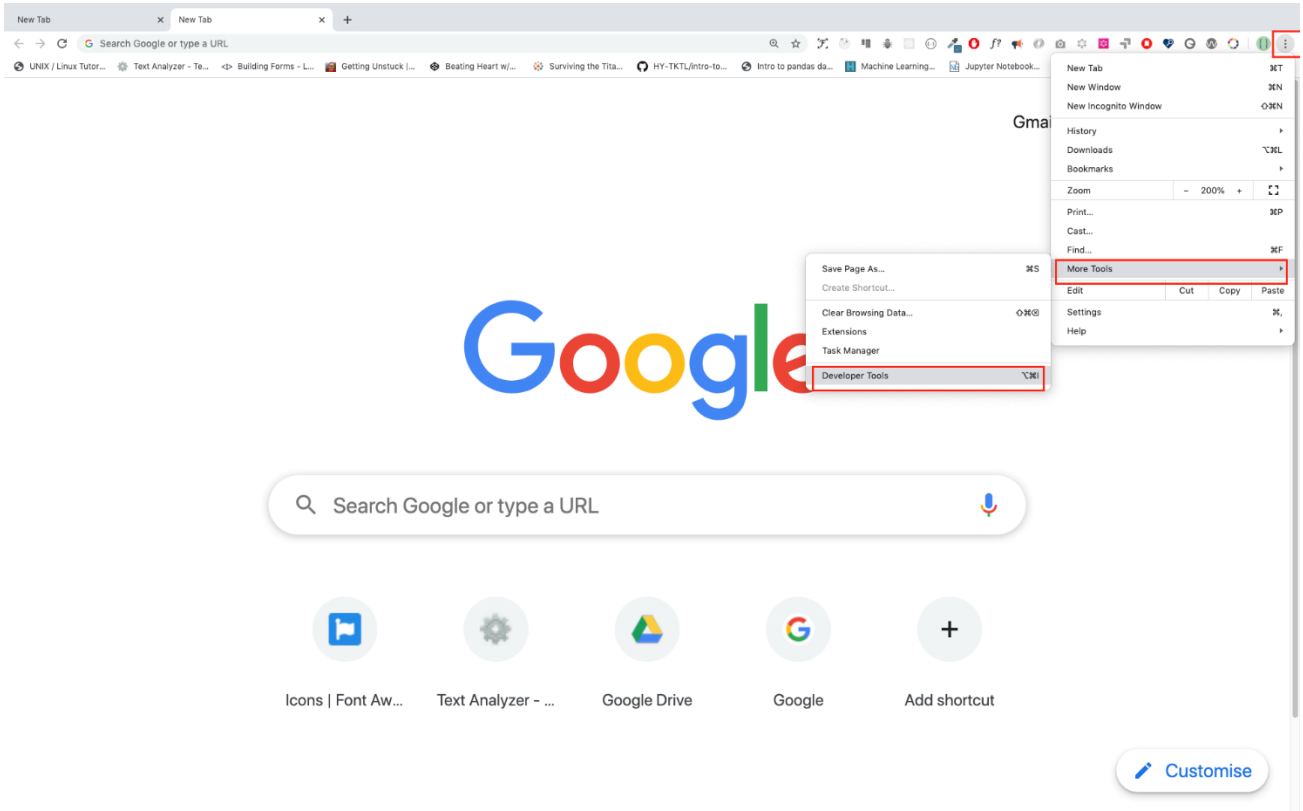
Installing Google Chrome

Install Google Chrome if you do not have one yet. We can write small JavaScript code on the browser console, but we do not use the browser console to develop applications.



Opening Google Chrome Console

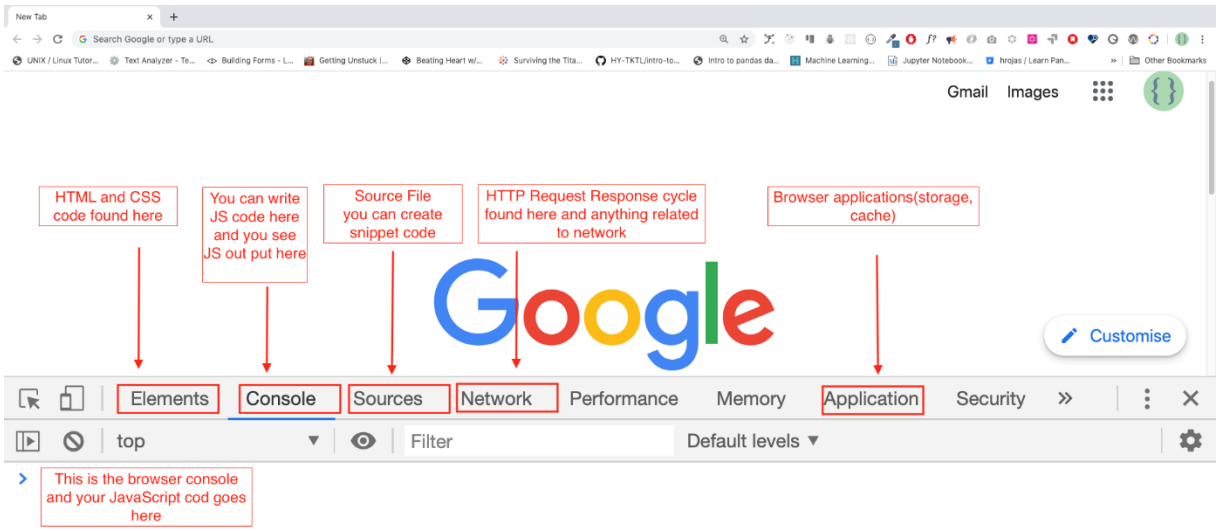
You can open Google Chrome console either by clicking three dots at the top right corner of the browser, selecting *More tools* -> *Developer tools* or using a keyboard shortcut. I prefer using shortcuts.



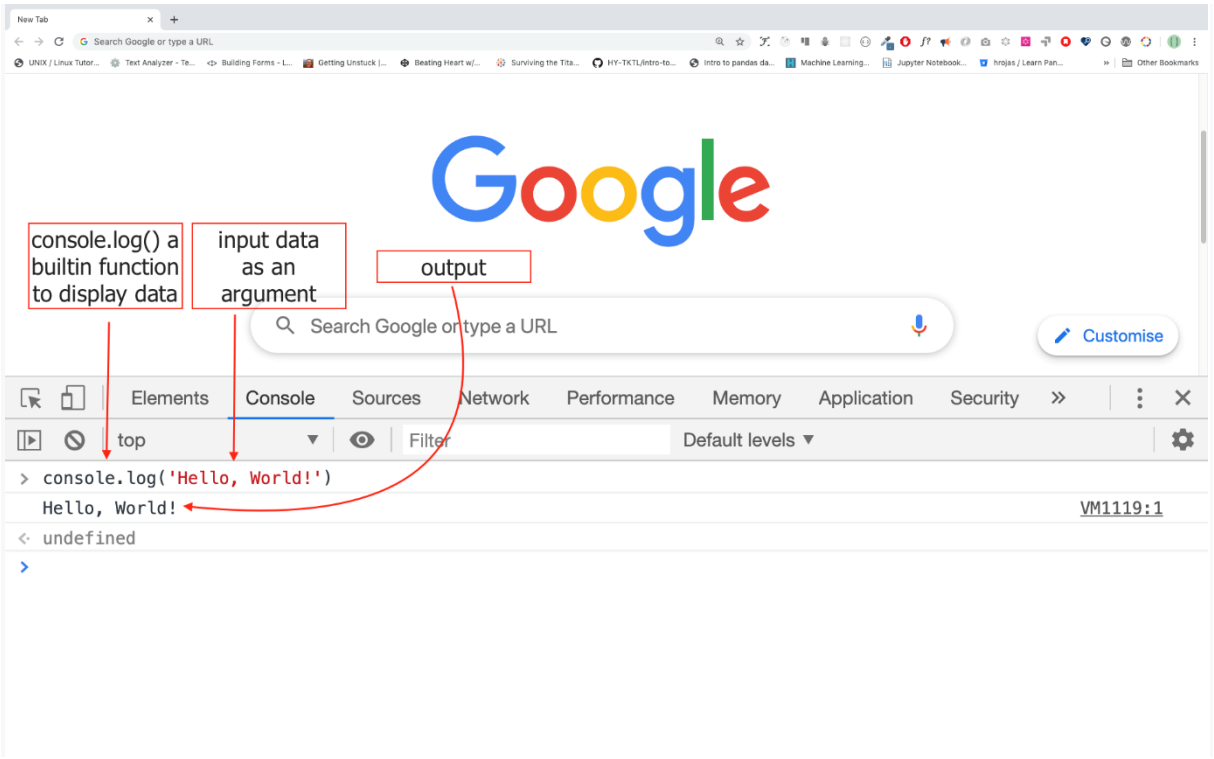
To open the Chrome console using a keyboard shortcut.

Mac
Command+Option+J

Windows/Linux:
Ctrl+Shift+J



After you open the Google Chrome console, try to explore the marked buttons. We will spend most of the time on the Console. The Console is the place where your JavaScript code goes. The Google Console V8 engine changes your JavaScript code to machine code. Let us write a JavaScript code on the Google Chrome console:



Writing Code on Browser Console

We can write any JavaScript code on the Google console or any browser console. However, for this challenge, we only focus on Google Chrome console. Open the console using:

Mac
Command+Option+I

Windows:
Ctl+Shift+I

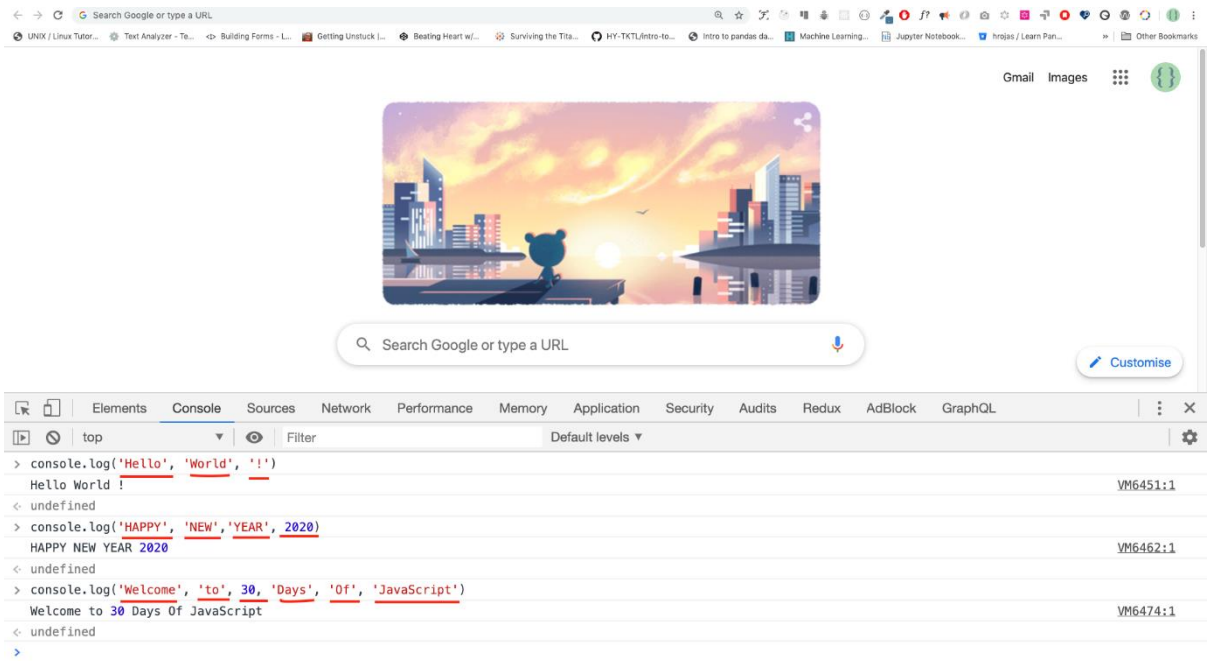
Console.log

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

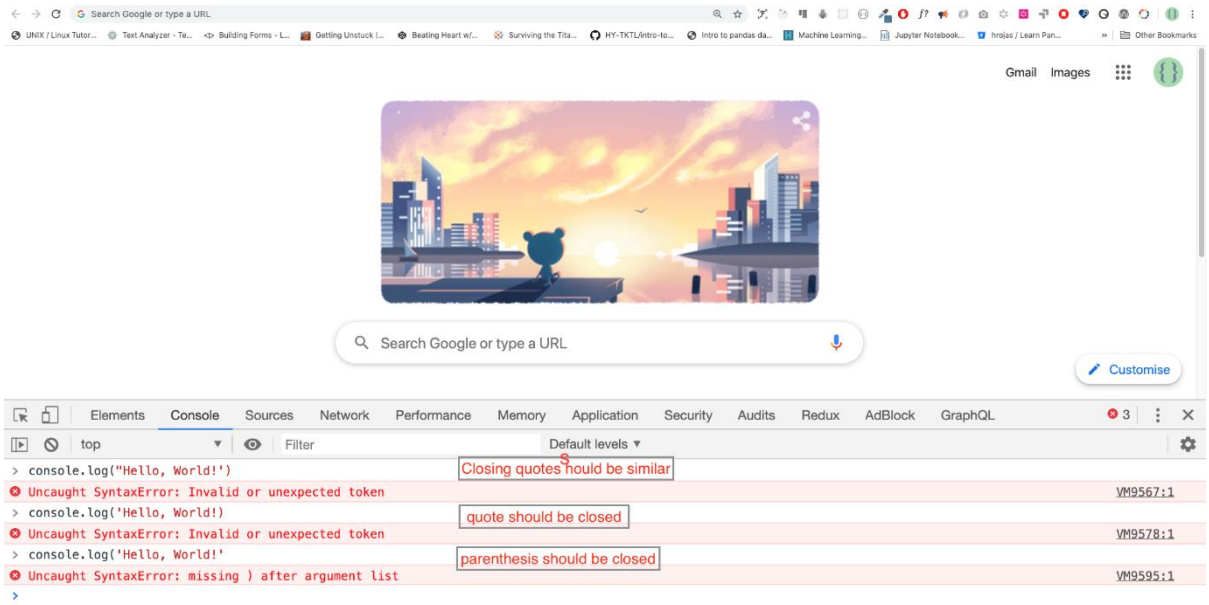
```
// This is the first comment
// This is the second comment
// I am a single line comment
```

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!')
```

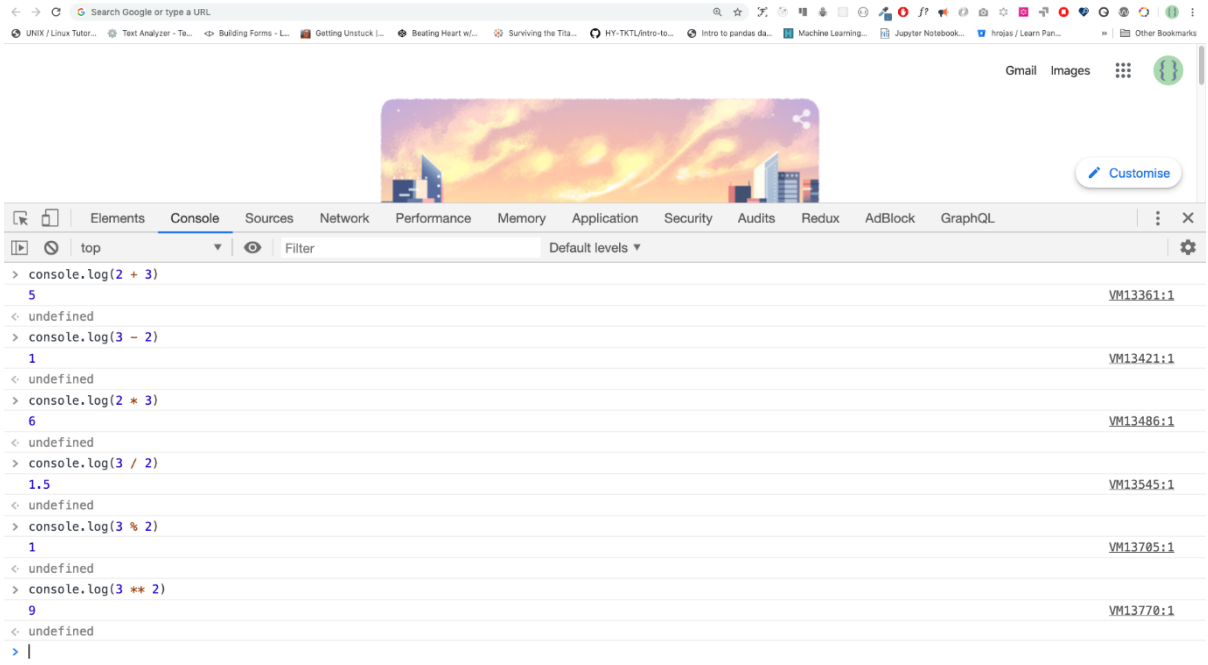
So far, we saw how to display text using the *console.log()*. If we are printing text or string using *console.log()*, the text has to be inside the single quotes, double quotes, or a backtick. **Example:**

```
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.



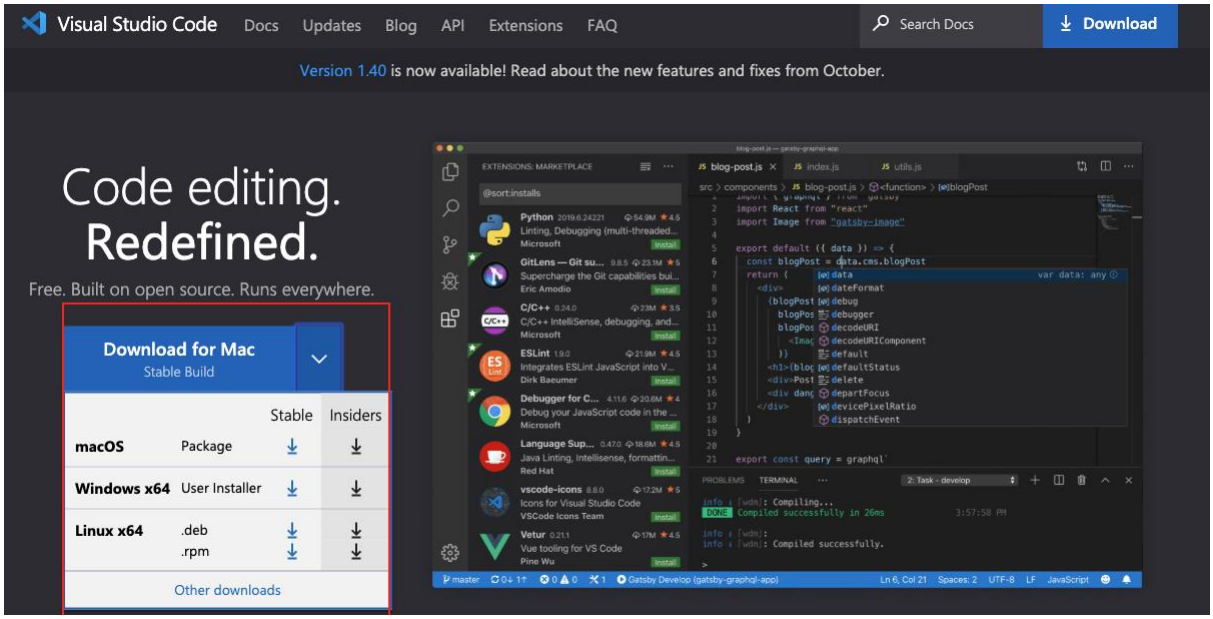

```
console.log(2 + 3) // Addition
console.log(3 - 2) // Subtraction
console.log(2 * 3) // Multiplication
console.log(3 / 2) // Division
console.log(3 % 2) // Modulus - finding remainder
console.log(3 ** 2) // Exponentiation 3 ** 2 == 3 * 3
```

Code Editor

We can write our codes on the browser console, but it won't be for bigger projects. In a real working environment, developers use different code editors to write their codes. In this 30 days of JavaScript challenge, we will be using Visual Studio Code.

Installing Visual Studio Code

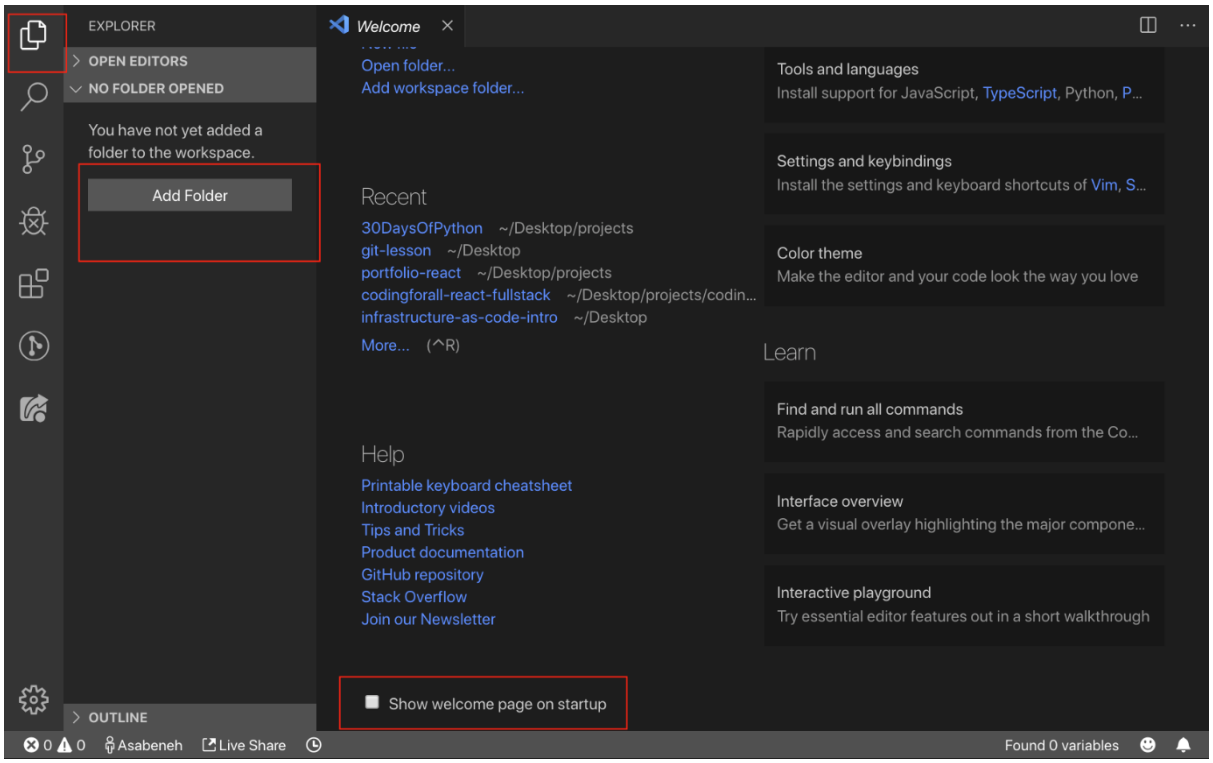
Visual Studio Code is a very popular open-source text editor. I would recommend to download Visual Studio Code, but if you are in favor of other editors, feel free to follow with what you have.

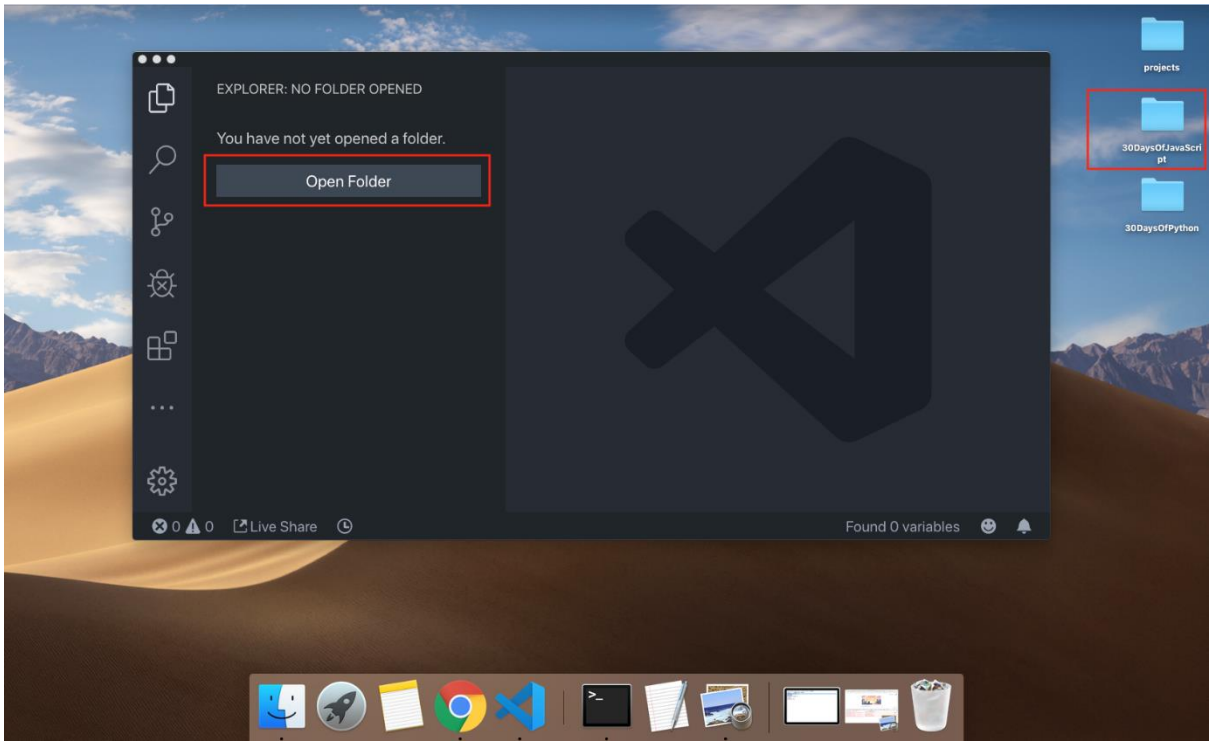


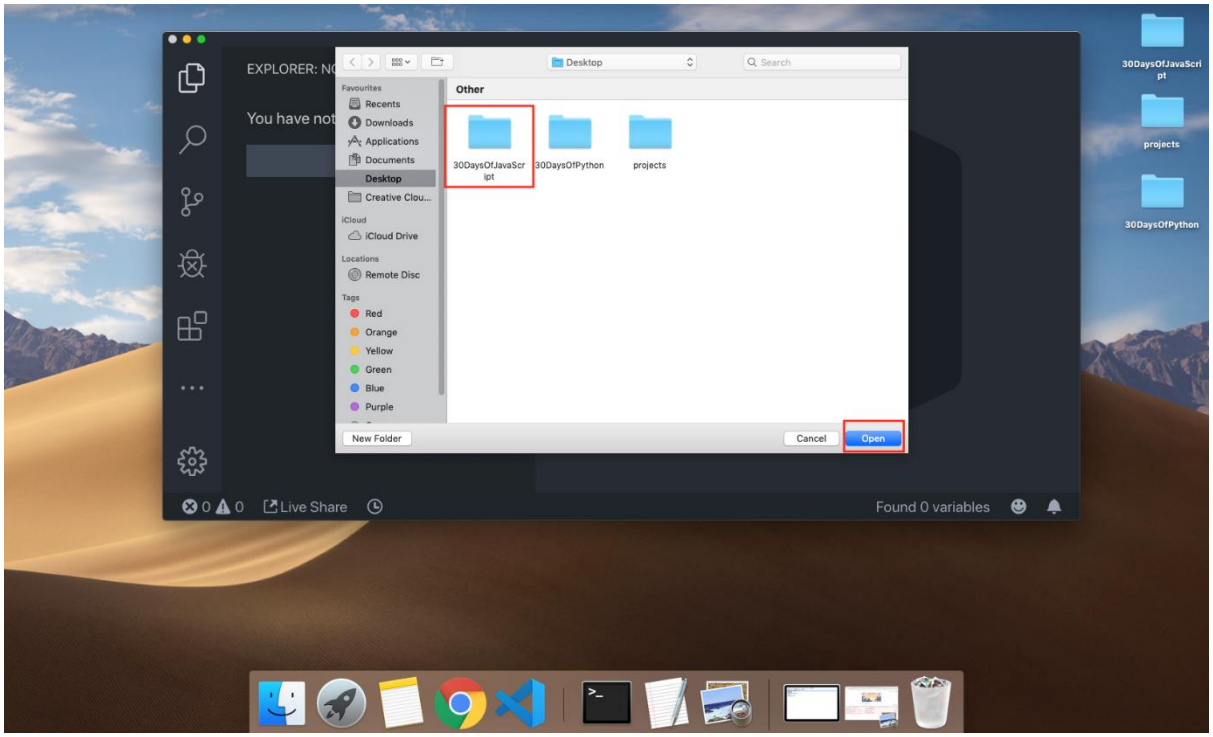
If you installed Visual Studio Code, let us start using it.

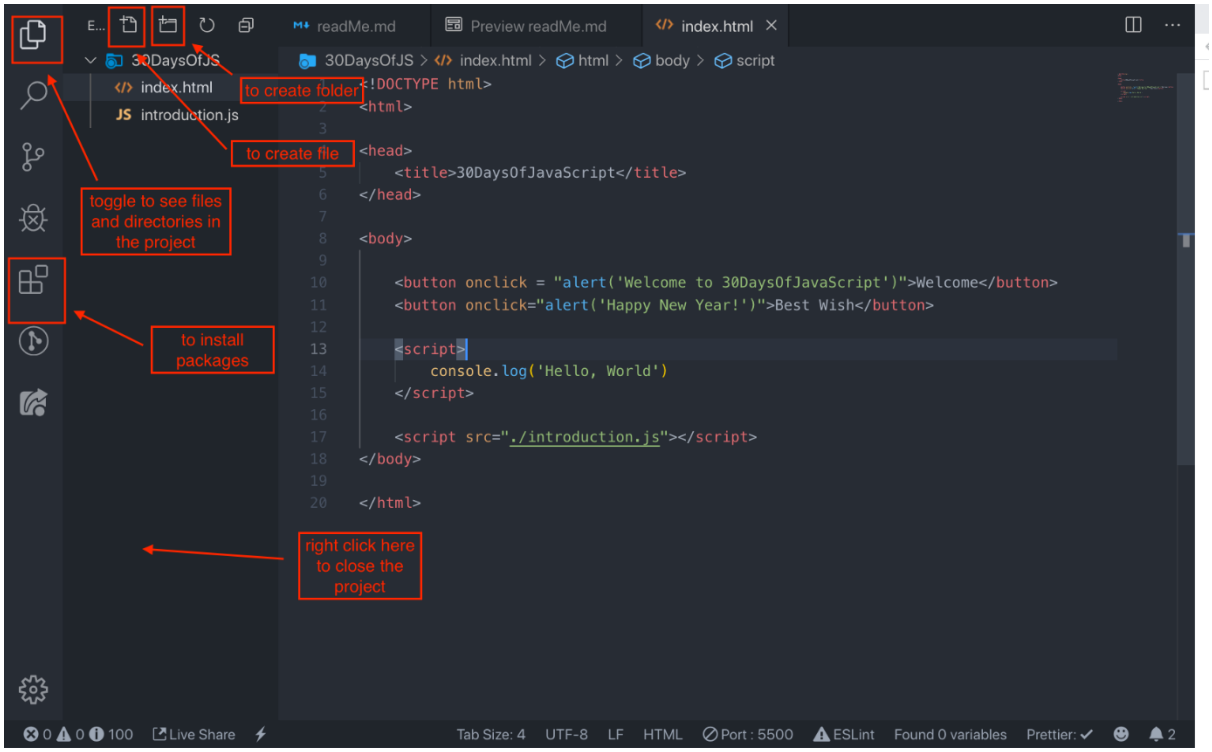
How to Use Visual Studio Code

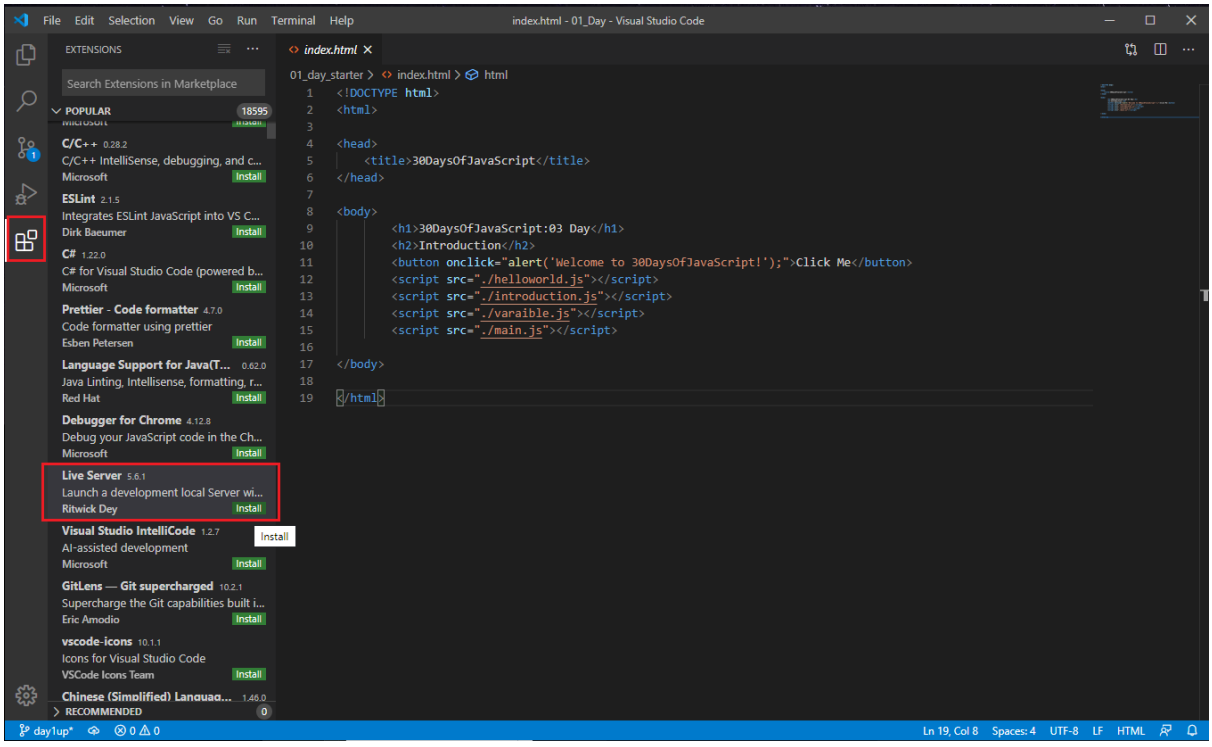
Open the Visual Studio Code by double-clicking its icon. When you open it, you will get this kind of interface. Try to interact with the labeled icons.

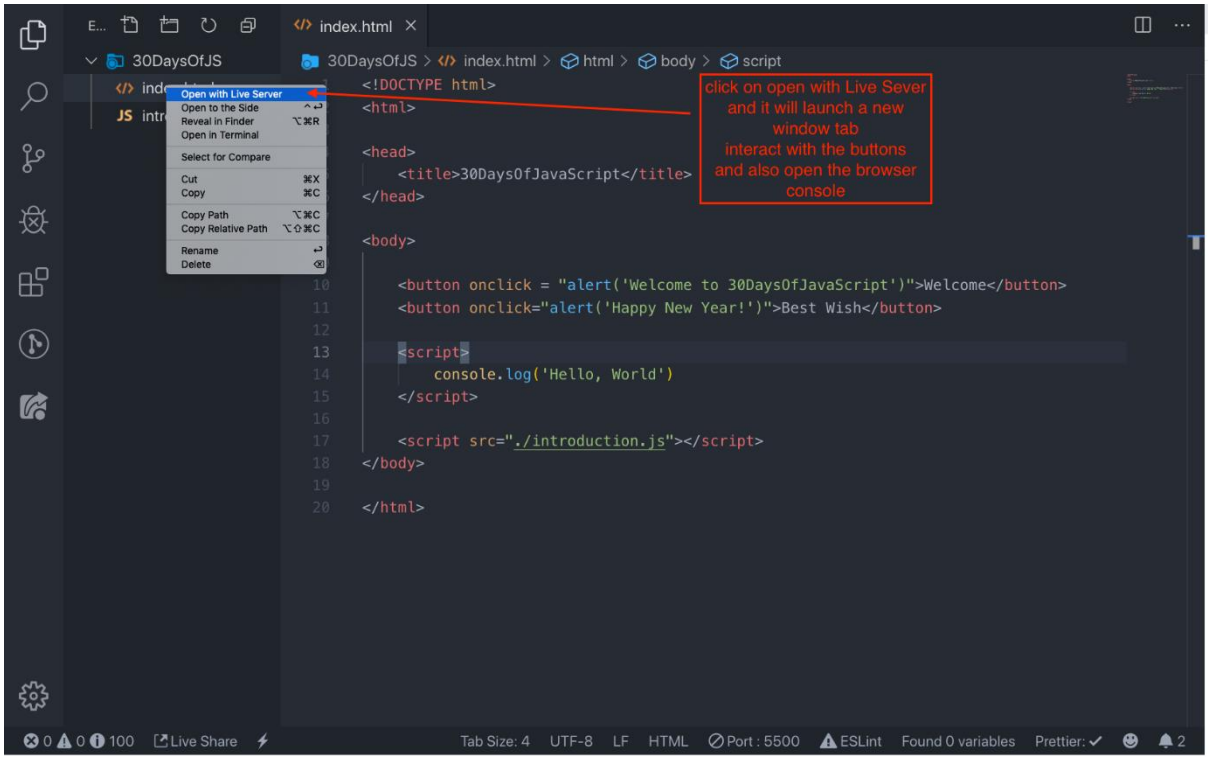


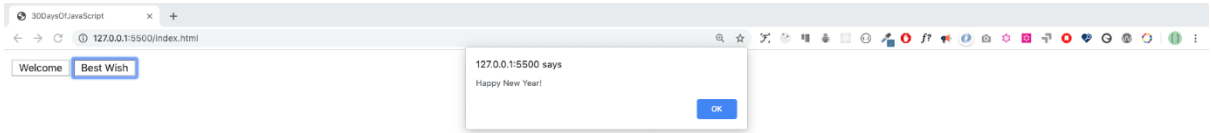


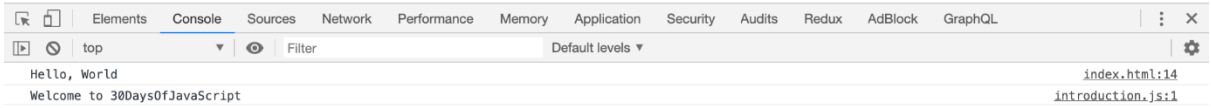












Adding JavaScript to a Web Page

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>30DaysOfScript:Inline Script</title>
  </head>
  <body>
    <button onclick="alert('Welcome to 30DaysOfJavaScript!')">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

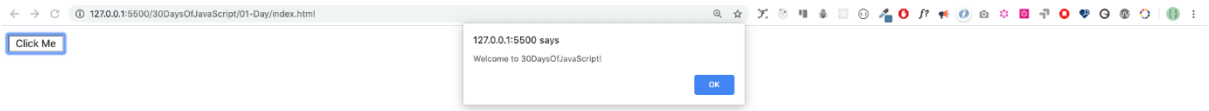
The internal script can be written in the *head* or the *body*, but it is preferred to put it on the body of the HTML document. First, let us write on the head part of the page.

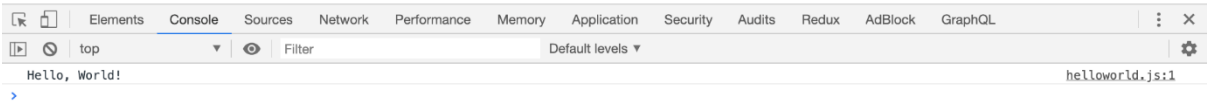
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>30DaysOfScript:Internal Script</title>
    <script>
      console.log('Welcome to 30DaysOfJavaScript')
    </script>
  </head>
  <body></body>
</html>
```

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>30DaysOfScript:Internal Script</title>
  </head>
  <body>
    <button onclick="alert('Welcome to 30DaysOfJavaScript!');">Click
Me</button>
    <script>
      console.log('Welcome to 30DaysOfJavaScript')
    </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 30DaysOfJavaScript')
```

External scripts in the *head*:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>30DaysOfJavaScript:External script</title>
    <script src="introduction.js"></script>
  </head>
  <body></body>
</html>
```

External scripts in the *body*:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>30DaysOfJavaScript: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().`

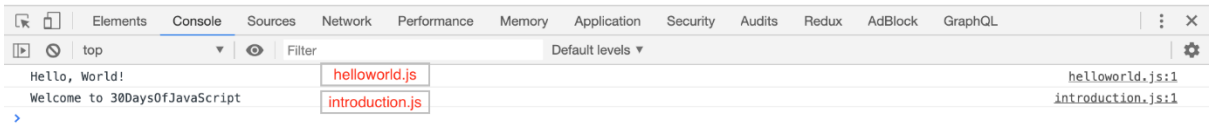
Multiple External Scripts

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'
'Asabeneh'
"Asabeneh"
'Finland'
'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 'Asabeneh') // 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*

Singleline commenting:

```
// commenting the code itself with a single comment
// let firstName = 'Asabeneh'; single line comment
// let lastName = 'Yetayeh'; 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
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 = 'Asabeneh' // first name of a person
let lastName = 'Yetayeh' // last name of a person
let country = 'Finland' // country
let city = 'Helsinki' // capital city
let age = 100 // age in years
let isMarried = true

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

```
Asabeneh Yetayeh Finland Helsinki 100 true
```

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

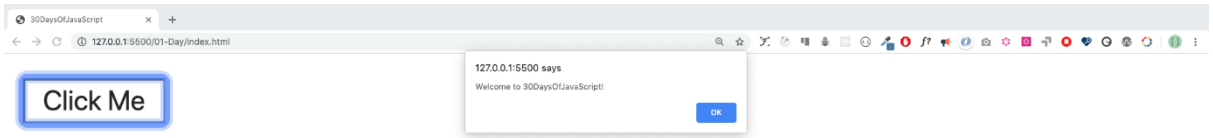
```
const boilingPoint = 100 // water boiling point, temperature in °C
const PI = 3.14 // geometrical constant
console.log(gravity, boilingPoint, PI)
```

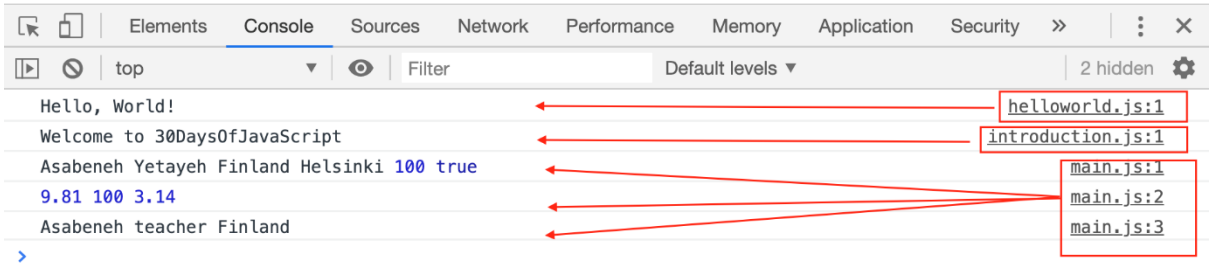
```
9.81 100 3.14
```

```
// 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 = 'Asabeneh', job = 'teacher', live = 'Finland'
console.log(name, job, live)
```

```
Asabeneh teacher Finland
```

When you run *index.html* file in the 01-Day folder you should get this:





You're off to a great start! Completing your first step in JavaScript is an achievement, and you're building a strong foundation. Now, challenge yourself with some exercises to reinforce your learning and sharpen your skills!

💻 Day 1: Exercises

1. Write a single line comment which says, *comments can make code readable*
2. Write another single comment which says, *Welcome to 30DaysOfJavaScript*
3. Write a multiline comment which says, *comments can make code readable, easy to reuse and informative*
4. Create a variable.js file and declare variables and assign string, boolean, undefined and null data types
5. Create datatypes.js file and use the JavaScript ***typeof*** operator to check different data types. Check the data type of each variable
6. Declare four variables without assigning values
7. Declare four variables with assigned values
8. Declare variables to store your first name, last name, marital status, country and age in multiple lines
9. Declare variables to store your first name, last name, marital status, country and age in a single line
10. Declare two variables *myAge* and *yourAge* and assign them initial values and log to the browser console.

```
I am 25 years old.
You are 30 years old.
```

🎉 CONGRATULATIONS ! DAY-1 SUCCESSFULLY COMPLETED 🎉

DAY-2 DATATYPES

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:

Primitive data types

Non-primitive data types(Object References)

Primitive Data Types

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:'Asabeneh',
role:'teaching',
country:'Finland'
}

let userTwo = {
name:'Asabeneh',
role:'teaching',
country:'Finland'
}

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:'Asabeneh',
role:'teaching',
country:'Finland'
}

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 = 'Asabeneh'
let lastName = 'Yetayeh'
let country = 'Finland'
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);
```

```
Asabeneh Yetayeh
```

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 = 'Asabeneh'
let lastName = 'Yetayeh'
let country = 'Finland'
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)
```

```
Asabeneh Yetayeh. I am 250. I live in Finland
```

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 Asabeneh Yetayeh. I live in Finland, 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
30DaysOfJavaScript 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

```
let firstName = 'Asabeneh'
let lastName = 'Yetayeh'
let country = 'Finland'
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 Asabeneh Yetayeh. I am 250. I live in Finland.
I am Asabeneh Yetayeh. I live in Helsinki, Finland. 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

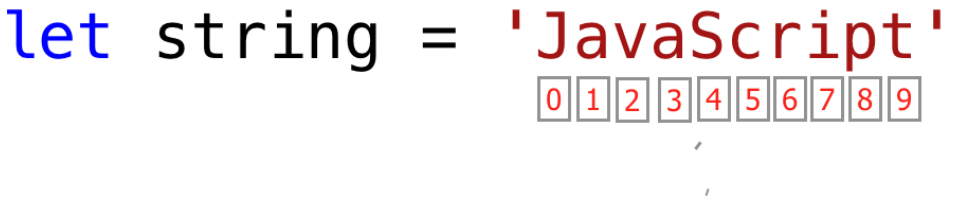
Everything in JavaScript is an object. A string is a primitive data type that means we can not modify it once it is created. The string object has many string methods. There are different string methods that can help us to work with strings.

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 = 'Asabeneh'
   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 = 'Asabeneh'

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

let country = 'Finland'

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

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

```
let string = 'JavasCript'

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

let firstName = 'Asabeneh'

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

let country = 'Finland'

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

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 = 'Finland'
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 = 'Finland'

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 = 'Asabeneh'

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

let countries = 'Finland, Sweden, Norway, Denmark, and
Iceland'

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

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
  Asabeneh
Asabeneh
```

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 = 'Finland'

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 = 'Finland'
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")) //
30DaysOfJavaScript

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

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 = 'Finland'

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 = 'Finland'

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.'
```

```
["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 = 'Asabeneh'      // string
let lastName = 'Yetayeh'        // string
let country = 'Finland'         // 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 'Asabeneh')  // 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
```

Great job! You've completed another step in your JavaScript journey, building momentum along the way. Keep going strong—now, challenge yourself with some exercises to reinforce your learning and sharpen your skills!

Day 2: Exercises

Exercise: Level 1

1. Declare a variable named challenge and assign it to an initial value **'30 Days Of JavaScript'**.
2. Print the string on the browser console using **console.log()**
3. Print the **length** of the string on the browser console using *console.log()*
4. Change all the string characters to capital letters
   using **toUpperCase()** method
5. Change all the string characters to lowercase letters
   using **toLowerCase()** method
6. Cut (slice) out the first word of the string
   using **substr()** or **substring()** method
7. Slice out the phrase *Days Of JavaScript* from *30 Days Of JavaScript*.
8. Check if the string contains a word **Script** using **includes()** method
9. Split the **string** into an **array** using **split()** method
10. Split the string 30 Days Of JavaScript at the space using **split()** method
11. 'Facebook, Google, Microsoft, Apple, IBM, Oracle, Amazon' **split** the string at the comma and change it to an array.
12. Change 30 Days Of JavaScript to 30 Days Of Python
    using **replace()** method.
13. What is character at index 15 in '30 Days Of JavaScript' string?
    Use **charAt()** method.
14. What is the character code of J in '30 Days Of JavaScript' string
    using **charCodeAt()**
15. Use **indexOf** to determine the position of the first occurrence of **a** in 30 Days Of JavaScript
16. Use **lastIndexOf** to determine the position of the last occurrence of **a** in 30 Days Of JavaScript.

17. Use **indexOf** to find the position of the first occurrence of the word **because** in the following sentence:**'You cannot end a sentence with because because because is a conjunction'**
18. Use **lastIndexOf** to find the position of the last occurrence of the word **because** in the following sentence:**'You cannot end a sentence with because because because is a conjunction'**
19. Use **search** to find the position of the first occurrence of the word **because** in the following sentence:**'You cannot end a sentence with because because because is a conjunction'**
20. Use **trim()** to remove any trailing whitespace at the beginning and the end of a string.E.g ' 30 Days Of JavaScript '.
21. Use **startsWith()** method with the string *30 Days Of JavaScript* and make the result true
22. Use **endsWith()** method with the string *30 Days Of JavaScript* and make the result true
23. Use **match()** method to find all the **a**'s in 30 Days Of JavaScript
24. Use **concat()** and merge '30 Days of' and 'JavaScript' to a single string, '30 Days Of JavaScript'
25. Use **repeat()** method to print 30 Days Of JavaScript 2 times

Exercise: Level 2

1. Using console.log() print out the following statement:

```
The quote 'There is no exercise better for the heart than reaching down and lifting people up.' by John Holmes teaches us to help one another.
```

2. Using console.log() print out the following quote by Mother Teresa:

```
"Love is not patronizing and charity isn't about pity, it is about love. Charity and love are the same -- with charity you give love, so don't just give money but reach out your hand instead."
```

3. Check if typeof '10' is exactly equal to 10. If not make it exactly equal.
4. Check if parseFloat('9.8') is equal to 10 if not make it exactly equal with 10.
5. Check if 'on' is found in both python and jargon
6. *I hope this course is not full of jargon*. Check if *jargon* is in the sentence.
7. Generate a random number between 0 and 100 inclusively.
8. Generate a random number between 50 and 100 inclusively.
9. Generate a random number between 0 and 255 inclusively.
10. Access the 'JavaScript' string characters using a random number.
11. Use console.log() and escape characters to print the following pattern.

```
1 1 1 1 1
2 1 2 4 8
3 1 3 9 27
4 1 4 16 64
     5  1 5 25 125
```

12. Use **substr** to slice out the phrase **because because because** from the following sentence:**'You cannot end a sentence with because because because is a conjunction'**

Exercises: Level 3

1. 'Love is the best thing in this world. Some found their love and some are still looking for their love.' Count the number of word **love** in this sentence.
2. Use **match()** to count the number of all **because** in the following sentence:**'You cannot end a sentence with because because because is a conjunction'**
3. Clean the following text and find the most frequent word (hint, use replace and regular expressions).

```
const sentence = '%I $am@% a %tea@cher%, &and& I
lo%#ve %te@a@ching%;. The@re $is no@th@ing; &as& mo@re
rewarding as educa@ting &and& @emp%o@weri@ng peo@ple. ;I
found tea@ching m%o@re interesting tha@n any ot#her
%jo@bs. %Do@es thi%s mo@tiv#ate yo@u to be a tea@cher!?
%Th#is 30#Days&OfJavaScript &is al@so $the $resu@lt of
&love& of tea&ching'
```

4. Calculate the total annual income of the person by extracting the numbers from the following text. 'He earns 5000 euro from salary per month, 10000 euro annual bonus, 15000 euro online courses per month.'

DAY-3 Booleans, OPERATORS & DATES

Boolean

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 = 'Asabeneh'
let country = 'Finland'
```

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
```

Operator Precedence

I would like to recommend you to read about operator precedence from this link

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 30DaysOfJavaScript')
```

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 c
lick on the dialog box
```

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

Date Object

Time is an important thing. We like to know the time a certain activity or event. In JavaScript current time and date is created using JavaScript Date Object. The

object we create using Date object provides many methods to work with date and time.The methods we use to get date and time information from a date object values are started with a word *get* because it provide the information. *getFullYear(), getMonth(), getDate(), getDay(), getHours(), getMinutes, getSeconds(), getMilliseconds(), getTime(), getDay()*

Method	Description	Examples
getFullYear()	Get the **year** as a four digit number (yyyy)	2020
getMonth()	Get the **month** as a number (0-11)	0
getDate()	Get the **day** as a number (1-31)	4
getHours()	Get the **hour** (0-23)	0
getMinutes()	Get the **minute** (0-59)	56
getSeconds()	Get the **second** (0-59)	41
getMilliseconds()	Get the **millisecond** (0-999)	341
getTime()	Get the time (milliseconds since January 1, 1970)	1578092201341
getDay()	Get the weekday as a number (0-6)	6

Creating a time object

Once we create time object. The time object will provide information about time. Let us create a time object

```
const now = new Date()
console.log(now) // Sat Jan 04 2020 00:56:41 GMT+0200 (Eastern
European Standard Time)
```

We have created a time object and we can access any date time information from the object using the get methods we have mentioned on the table.

Getting full year

Let's extract or get the full year from a time object.

```
const now = new Date()
console.log(now.getFullYear()) // 2020
```

Getting month

Let's extract or get the month from a time object.

```
const now = new Date()
console.log(now.getMonth()) // 0, because the month is
January,  month(0-11)
```

Getting date

Let's extract or get the date of the month from a time object.

```
const now = new Date()
console.log(now.getDate()) // 4, because the day of the month
is 4th,  day(1-31)
```

Getting day

Let's extract or get the day of the week from a time object.

```
const now = new Date()
console.log(now.getDay()) // 6, because the day is Saturday
which is the 7th day
//  Sunday is 0, Monday is 1 and Saturday is 6
// Getting the weekday as a number (0-6)
```

Getting hours

Let's extract or get the hours from a time object.

```
const now = new Date()
console.log(now.getHours()) // 0, because the time is 00:56:41
```

Getting minutes

Let's extract or get the minutes from a time object.

```
const now = new Date()
console.log(now.getMinutes()) // 56, because the time is
00:56:41
```

Getting seconds

Let's extract or get the seconds from a time object.

```
const now = new Date()
console.log(now.getSeconds()) // 41, because the time is
00:56:41
```

Getting time

This method give time in milliseconds starting from January 1, 1970. It is also know as Unix time. We can get the unix time in two ways:

1. Using *getTime()*

```
const now = new Date() //
console.log(now.getTime()) // 1578092201341, this is the
number of seconds passed from January 1, 1970 to January 4,
2020 00:56:41
```

2. Using *Date.now()*

```
const allSeconds = Date.now() //
console.log(allSeconds) // 1578092201341, this is the number
of seconds passed from January 1, 1970 to January 4, 2020
00:56:41

const timeInSeconds = new Date().getTime()
console.log(allSeconds == timeInSeconds) // true
```

Let us format these values to a human readable time format. **Example:**

```
const now = new Date()
const year = now.getFullYear() // return year
const month = now.getMonth() + 1 // return month(0 - 11)
const date = now.getDate() // return date (1 - 31)
const hours = now.getHours() // return number (0 - 23)
const minutes = now.getMinutes() // return number (0 -59)

console.log(`${date}/${month}/${year} ${hours}:${minutes}`) //
4/1/2020 0:56
```

 You have boundless energy. You have just completed day 3 challenges and you are three steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Day 3: Exercises

Exercises: Level 1

1. Declare firstName, lastName, country, city, age, isMarried, year variable and assign value to it and use the typeof operator to check different data types.
2. Check if type of '10' is equal to 10
3. Check if parseInt('9.8') is equal to 10
4. Boolean value is either true or false.
    i. Write three JavaScript statement which provide truthy value.
    ii. Write three JavaScript statement which provide falsy value.
5. Figure out the result of the following comparison expression first without using console.log(). After you decide the result confirm it using console.log()
    i. 4 > 3
    ii. 4 >= 3
    iii. 4 < 3
    iv. 4 <= 3
    v. 4 == 4
    vi. 4 === 4
    vii. 4 != 4
    viii. 4 !== 4
    ix. 4 != '4'
    x. 4 == '4'
    xi. 4 === '4'
    xii. Find the length of python and jargon and make a falsy comparison statement.
6. Figure out the result of the following expressions first without using console.log(). After you decide the result confirm it by using console.log()
    i. 4 > 3 && 10 < 12

ii. 4 > 3 && 10 > 12

iii. 4 > 3 || 10 < 12

iv. 4 > 3 || 10 > 12

v. !(4 > 3)

vi. !(4 < 3)

vii. !(false)

viii. !(4 > 3 && 10 < 12)

ix. !(4 > 3 && 10 > 12)

x. !(4 === '4')

xi. There is no 'on' in both dragon and python

7. Use the Date object to do the following activities
   i. What is the year today?
   ii. What is the month today as a number?
   iii. What is the date today?
   iv. What is the day today as a number?
   v. What is the hours now?
   vi. What is the minutes now?
   vii. Find out the numbers of seconds elapsed from January 1, 1970 to now.

Exercises: Level 2

1. Write a script that prompt the user to enter base and height of the triangle and calculate an area of a triangle (area = 0.5 x b x h).

```
Enter base: 20
Enter height: 10
The area of the triangle is 100
```

2. Write a script that prompt the user to enter side a, side b, and side c of the triangle and and calculate the perimeter of triangle (perimeter = a + b + c)

```
Enter side a: 5
Enter side b: 4
Enter side c: 3
```

```
The perimeter of the triangle is 12
```

3. Get length and width using prompt and calculate an area of rectangle (area = length x width and the perimeter of rectangle (perimeter = 2 x (length + width))
4. Get radius using prompt and calculate the area of a circle (area = pi x r x r) and circumference of a circle(c = 2 x pi x r) where pi = 3.14.
5. Calculate the slope, x-intercept and y-intercept of y = 2x -2
6. Slope is m = $(y_2-y_1)/(x_2-x_1)$. Find the slope between point (2, 2) and point(6,10)
7. Compare the slope of above two questions.
8. Calculate the value of y ($y = x^2 + 6x + 9$). Try to use different x values and figure out at what x value y is 0.
9. Writ a script that prompt a user to enter hours and rate per hour. Calculate pay of the person?

```
Enter hours: 40
Enter rate per hour: 28
Your weekly earning is 1120
```

10. If the length of your name is greater than 7 say, your name is long else say your name is short.
11. Compare your first name length and your family name length and you should get this output.

```
let firstName = 'Asabeneh'
let lastName = 'Yetayeh'
```

```
Your first name, Asabeneh is longer than your family
name, Yetayeh
```

12. Declare two variables *myAge* and *yourAge* and assign them initial values and myAge and yourAge.

```
let myAge = 250
let yourAge = 25
```

```
I am 225 years older than you.
```

13. Using prompt get the year the user was born and if the user is 18 or above allow the user to drive if not tell the user to wait a certain amount of years.

```
Enter birth year: 1995
You are 25. You are old enough to drive
```

```
Enter birth year: 2005
You are 15. You will be allowed to drive after 3 years.
```

14. Write a script that prompt the user to enter number of years. Calculate the number of seconds a person can live. Assume some one lives just hundred years

```
Enter number of years you live: 100
You lived 3153600000 seconds.
```

15. Create a human readable time format using the Date time object
    i. YYYY-MM-DD HH:mm
    ii. DD-MM-YYYY HH:mm
    iii. DD/MM/YYYY HH:mm

Exercises: Level 3

1. Create a human readable time format using the Date time object. The hour and the minute should be all the time two digits(7 hours should be 07 and 5 minutes should be 05)
    i. YYY-MM-DD HH:mm eg. 20120-01-02 07:05

DAY-4 CONDITIONALS

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.')
```

You are extraordinary and you have a remarkable potential. You have just completed day 4 challenges and you are four steps ahead to your way to greatness. Now do some exercises for your brain and muscle.

💻 Exercises

Exercises: Level 1

1. Get user input using prompt("Enter your age:"). If user is 18 or older , give feedback:'You are old enough to drive' but if not 18 give another feedback stating to wait for the number of years he needs to turn 18.

```
Enter your age: 30
You are old enough to drive.

Enter your age:15
You are left with 3 years to drive.
```

2. Compare the values of myAge and yourAge using if … else. Based on the comparison and log the result to console stating who is older (me or you). Use prompt("Enter your age:") to get the age as input.

```
Enter your age: 30
You are 5 years older than me.
```

3. If a is greater than b return 'a is greater than b' else 'a is less than b'. Try to implement it in to ways
    - using if else
    - ternary operator.

```
let a = 4
let b = 3
```

```
4 is greater than 3
```

4. Even numbers are divisible by 2 and the remainder is zero. How do you check, if a number is even or not using JavaScript?

```
Enter a number: 2
2 is an even number

Enter a number: 9
9 is is an odd number.
```

Exercises: Level 2

1. Write a code which can give grades to students according to theirs scores:
    - 80-100, A
    - 70-89, B
    - 60-69, C
    - 50-59, D
    - 0-49, F
2. Check if the season is Autumn, Winter, Spring or Summer. If the user input is :
    - September, October or November, the season is Autumn.
    - December, January or February, the season is Winter.
    - March, April or May, the season is Spring
    - June, July or August, the season is Summer
3. Check if a day is weekend day or a working day. Your script will take day as an input.

```
What is the day  today? Saturday
Saturday is a weekend.

What is the day today? saturDaY
Saturday is a weekend.

What is the day today? Friday
Friday is a working day.

What is the day today? FrIDAy
Friday is a working day.
```

Exercises: Level 3

1. Write a program which tells the number of days in a month.

```
Enter a month: January
January has 31 days.

Enter a month: JANUARY
January has 31 day

Enter a month: February
February has 28 days.

Enter a month: FEbruary
February has 28 days.
```

2. Write a program which tells the number of days in a month, now consider leap year.

CONGRATULATIONS ! YOU SUCCESSFULLY COMPLETED DAY 3 🎉

DAY-5 ARRAYS

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 = ['Finland', '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: ['Finland', 'Estonia', 'Denmark', 'Sweden',
'Norway']
Number of countries: 5
```

- Array can have items of different data types

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

Creating an array using split

As we have seen in the earlier section, we can split a string at different positions, and we can change to an array. Let us see the examples below.

```
let js = 'JavaScript'
const charsInJavaScript = js.split('')

console.log(charsInJavaScript) // ["J", "a", "v", "a", "S",
"c", "r", "i", "p", "t"]

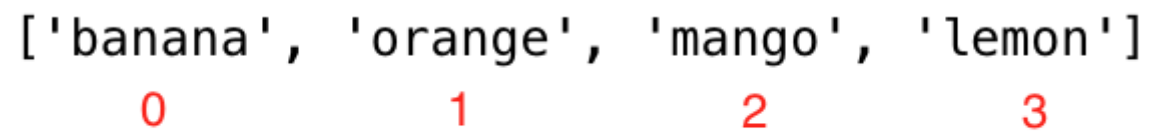
let companiesString = 'Facebook, Google, Microsoft, Apple,
IBM, Oracle, Amazon'
const companies = companiesString.split(',')

console.log(companies) // ["Facebook", " Google", "
Microsoft", " Apple", " IBM", " Oracle", " Amazon"]
let txt =
  'I love teaching and empowering people. I teach HTML, CSS,
JS, React, Python.'
const words = txt.split(' ')

console.log(words)
// the text has special characters think how you can just get
only the words
// ["I", "love", "teaching", "and", "empowering", "people.",
"I", "teach", "HTML,", "CSS,", "JS,", "React,", "Python"]
```

Accessing array items using index

We access each element in an array using their index. An array index starts from 0. The picture below clearly shows the index of each element in the array.



```
const fruits = ['banana', 'orange', 'mango', 'lemon']
let firstFruit = fruits[0] // we are accessing the first item
using its index

console.log(firstFruit) // banana

secondFruit = fruits[1]
console.log(secondFruit) // orange

let lastFruit = fruits[3]
console.log(lastFruit) // lemon
// Last index can be calculated as follows

let lastIndex = fruits.length - 1
lastFruit = fruits[lastIndex]

console.log(lastFruit)  // lemon
```

```
const numbers = [0, 3.14, 9.81, 37, 98.6, 100]  // set of
numbers

console.log(numbers.length)  // => to know the size of the
array, which is 6
console.log(numbers)         // -> [0, 3.14, 9.81, 37, 98.6,
100]
console.log(numbers[0])      //  -> 0
console.log(numbers[5])      //  -> 100

let lastIndex = numbers.length - 1;
console.log(numbers[lastIndex]) // -> 100
```

```
const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
  'React',
  'Redux',
  'Node',
  'MongoDB'
] // List of web technologies

console.log(webTechs)        // all the array items
console.log(webTechs.length) // => to know the size of the
array, which is 7
console.log(webTechs[0])     //  -> HTML
console.log(webTechs[6])     //  -> MongoDB
```

```
let lastIndex = webTechs.length - 1
console.log(webTechs[lastIndex]) // -> MongoDB
const countries = [
  'Albania',
  'Bolivia',
  'Canada',
  'Denmark',
  'Ethiopia',
  'Finland',
  'Germany',
  'Hungary',
  'Ireland',
  'Japan',
  'Kenya'
] // List of countries

console.log(countries)      // -> all countries in array
console.log(countries[0])   //  -> Albania
console.log(countries[10])  //  -> Kenya

let lastIndex = countries.length - 1;
console.log(countries[lastIndex]) //  -> Kenya
```

```
const shoppingCart = [
  'Milk',
  'Mango',
  'Tomato',
  'Potato',
  'Avocado',
  'Meat',
  'Eggs',
  'Sugar'
] // List of food products

console.log(shoppingCart) // -> all shoppingCart in array
console.log(shoppingCart[0]) //  -> Milk
console.log(shoppingCart[7]) //  -> Sugar

let lastIndex = shoppingCart.length - 1;
console.log(shoppingCart[lastIndex]) //  -> Sugar
```

Modifying array element

An array is mutable(modifiable). Once an array is created, we can modify the contents of the array elements.

```
const numbers = [1, 2, 3, 4, 5]
numbers[0] = 10      // changing 1 at index 0 to 10
numbers[1] = 20      // changing  2 at index 1 to 20

console.log(numbers) // [10, 20, 3, 4, 5]

const countries = [
  'Albania',
  'Bolivia',
  'Canada',
  'Denmark',
  'Ethiopia',
  'Finland',
  'Germany',
  'Hungary',
  'Ireland',
  'Japan',
  'Kenya'
]

countries[0] = 'Afghanistan'  // Replacing Albania by
Afghanistan
let lastIndex = countries.length - 1
countries[lastIndex] = 'Korea' // Replacing Kenya by Korea

console.log(countries)
```

```
["Afghanistan", "Bolivia", "Canada", "Denmark", "Ethiopia",
"Finland", "Germany", "Hungary", "Ireland", "Japan", "Korea"]
```

Methods to manipulate array

There are different methods to manipulate an array. These are some of the available methods to deal with arrays:*Array, length, concat, indexOf, slice, splice, join, toString, includes, lastIndexOf, isArray, fill, push, pop, shift, unshift*

Array Constructor

Array:To create an array.

```
const arr = Array() // creates an an empty array
console.log(arr)

const eightEmptyValues = Array(8) // it creates eight empty
values
console.log(eightEmptyValues) // [empty x 8]
```

Creating static values with fill

fill: Fill all the array elements with a static value

```
const arr = Array() // creates an an empty array
console.log(arr)

const eightXvalues = Array(8).fill('X') // it creates eight
element values filled with 'X'
console.log(eightXvalues) // ['X',
'X','X','X','X','X','X','X']

const eight0values = Array(8).fill(0) // it creates eight
element values filled with '0'
console.log(eight0values) // [0, 0, 0, 0, 0, 0, 0, 0]

const four4values = Array(4).fill(4) // it creates 4 element
values filled with '4'
console.log(four4values) // [4, 4, 4, 4]
```

Concatenating array using concat

concat:To concatenate two arrays.

```
const firstList = [1, 2, 3]
const secondList = [4, 5, 6]
const thirdList = firstList.concat(secondList)

console.log(thirdList) // [1, 2, 3, 4, 5, 6]
```

```
const fruits = ['banana', 'orange', 'mango', 'lemon']
// array of fruits
const vegetables = ['Tomato', 'Potato', 'Cabbage', 'Onion',
'Carrot'] // array of vegetables
const fruitsAndVegetables = fruits.concat(vegetables)
// concatenate the two arrays

console.log(fruitsAndVegetables)
```

```
["banana", "orange", "mango", "lemon", "Tomato", "Potato",
"Cabbage", "Onion", "Carrot"]
```

Getting array length

Length:To know the size of the array

```
const numbers = [1, 2, 3, 4, 5]
console.log(numbers.length) // -> 5 is the size of the array
```

Getting index an element in arr array

indexOf:To check if an item exist in an array. If it exists it returns the index else it returns -1.

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

console.log(numbers.indexOf(5)) // -> 4
console.log(numbers.indexOf(0)) // -> -1
console.log(numbers.indexOf(1)) // -> 0
console.log(numbers.indexOf(6)) // -> -1
```

Check an element if it exist in an array.

- Check items in a list

```
// let us check if a banana exist in the array

const fruits = ['banana', 'orange', 'mango', 'lemon']
let index = fruits.indexOf('banana')  // 0

if(index === -1){
   console.log('This fruit does not exist in the array')
} else {
    console.log('This fruit does exist in the array')
}
```

```
// This fruit does exist in the array

// we can use also ternary here
index === -1 ? console.log('This fruit does not exist in the
array'): console.log('This fruit does exist in the array')

// let us check if an avocado exist in the array
let indexOfAvocado = fruits.indexOf('avocado')  // -1, if the
element not found index is -1
if(indexOfAvocado === -1){
   console.log('This fruit does not exist in the array')
} else {
    console.log('This fruit does exist in the array')
}
// This fruit does not exist in the array
```

Getting last index of an element in array

lastIndexOf: It gives the position of the last item in the array. If it exist, it returns the index else it returns -1.

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

console.log(numbers.lastIndexOf(2)) // 7
console.log(numbers.lastIndexOf(0)) // -1
console.log(numbers.lastIndexOf(1)) //  6
console.log(numbers.lastIndexOf(4)) //  3
console.log(numbers.lastIndexOf(6)) // -1
```

includes:To check if an item exist in an array. If it exist it returns the true else it returns false.

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

console.log(numbers.includes(5)) // true
console.log(numbers.includes(0)) // false
console.log(numbers.includes(1)) // true
console.log(numbers.includes(6)) // false

const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
  'React',
  'Redux',
  'Node',
  'MongoDB'
] // List of web technologies
```

```
console.log(webTechs.includes('Node'))  // true
console.log(webTechs.includes('C'))     // false
```

Checking array

Array.isArray:To check if the data type is an array

```
const numbers = [1, 2, 3, 4, 5]
console.log(Array.isArray(numbers)) // true

const number = 100
console.log(Array.isArray(number)) // false
```

Converting array to string

toString:Converts array to string

```
const numbers = [1, 2, 3, 4, 5]
console.log(numbers.toString()) // 1,2,3,4,5

const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
console.log(names.toString()) // Asabeneh,Mathias,Elias,Brook
```

Joining array elements

join: It is used to join the elements of the array, the argument we passed in the join method will be joined in the array and return as a string. By default, it joins with a comma, but we can pass different string parameter which can be joined between the items.

```
const numbers = [1, 2, 3, 4, 5]
console.log(numbers.join()) // 1,2,3,4,5

const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']

console.log(names.join()) // Asabeneh,Mathias,Elias,Brook
console.log(names.join('')) //AsabenehMathiasEliasBrook
console.log(names.join(' ')) //Asabeneh Mathias Elias Brook
console.log(names.join(', ')) //Asabeneh, Mathias, Elias,
Brook
console.log(names.join(' # ')) //Asabeneh # Mathias # Elias #
Brook

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

```
  'Redux',
  'Node',
  'MongoDB'
] // List of web technologies

console.log(webTechs.join())       //
"HTML,CSS,JavaScript,React,Redux,Node,MongoDB"
console.log(webTechs.join(' # '))  // "HTML # CSS # JavaScript
# React # Redux # Node # MongoDB"
```

Slice array elements

Slice: To cut out a multiple items in range. It takes two parameters:starting and ending position. It doesn't include the ending position.

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

  console.log(numbers.slice()) // -> it copies all  item
  console.log(numbers.slice(0)) // -> it copies all  item
  console.log(numbers.slice(0, numbers.length)) // it copies
all  item
  console.log(numbers.slice(1,4)) // -> [2,3,4] // it doesn't
include the ending position
```

Splice method in array

Splice: It takes three parameters:Starting position, number of times to be removed and number of items to be added.

```
  const numbers = [1, 2, 3, 4, 5]
  numbers.splice()
  console.log(numbers)                // -> remove all items
```

```
  const numbers = [1, 2, 3, 4, 5]
     numbers.splice(0,1)
  console.log(numbers)            // remove the first item
```

```
 const numbers = [1, 2, 3, 4, 5, 6]
     numbers.splice(3, 3, 7, 8, 9)
  console.log(numbers.splice(3, 3, 7, 8, 9))  // -> [1, 2, 3,
7, 8, 9] //it removes three item and replace three items
```

Adding item to an array using push

Push: adding item in the end. To add item to the end of an existing array we use the push method.

```
// syntax
const arr  = ['item1', 'item2','item3']
arr.push('new item')
console.log(arr)
// ['item1', 'item2','item3','new item']
```

```
const numbers = [1, 2, 3, 4, 5]
numbers.push(6)
console.log(numbers) // -> [1,2,3,4,5,6]

numbers.pop() // -> remove one item from the end
console.log(numbers) // -> [1,2,3,4,5]
```

```
let fruits = ['banana', 'orange', 'mango', 'lemon']
fruits.push('apple')
console.log(fruits)    // ['banana', 'orange', 'mango',
'lemon', 'apple']

fruits.push('lime')
console.log(fruits)   // ['banana', 'orange', 'mango',
'lemon', 'apple', 'lime']
```

Removing the end element using pop

pop: Removing item in the end.

```
const numbers = [1, 2, 3, 4, 5]
numbers.pop() // -> remove one item from the end
console.log(numbers) // -> [1,2,3,4]
```

Removing an element from the beginning

shift: Removing one array element in the beginning of the array.

```
const numbers = [1, 2, 3, 4, 5]
numbers.shift() // -> remove one item from the beginning
console.log(numbers) // -> [2,3,4,5]
```

Add an element from the beginning

unshift: Adding array element in the beginning of the array.

```
const numbers = [1, 2, 3, 4, 5]
numbers.unshift(0) // -> add one item from the beginning
console.log(numbers) // -> [0,1,2,3,4,5]
```

Reversing array order

reverse: reverse the order of an array.

```
const numbers = [1, 2, 3, 4, 5]
numbers.reverse() // -> reverse array order
console.log(numbers) // [5, 4, 3, 2, 1]

numbers.reverse()
console.log(numbers) // [1, 2, 3, 4, 5]
```

Sorting elements in array

sort: arrange array elements in ascending order. Sort takes a call back function, we will see how we use sort with a call back function in the coming sections.

```
const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
  'React',
  'Redux',
  'Node',
  'MongoDB'
]

webTechs.sort()
console.log(webTechs) // ["CSS", "HTML", "JavaScript",
"MongoDB", "Node", "React", "Redux"]

webTechs.reverse() // after sorting we can reverse it
console.log(webTechs) // ["Redux", "React", "Node", "MongoDB",
"JavaScript", "HTML", "CSS"]
```

Array of arrays

Array can store different data types including an array itself. Let us create an array of arrays

```
const firstNums = [1, 2, 3]
const secondNums = [1, 4, 9]

const arrayOfArray =  [[1, 2, 3], [1, 2, 3]]
console.log(arrayOfArray[0]) // [1, 2, 3]

 const frontEnd = ['HTML', 'CSS', 'JS', 'React', 'Redux']
 const backEnd = ['Node','Express', 'MongoDB']
 const fullStack = [frontEnd, backEnd]
 console.log(fullStack)   // [["HTML", "CSS", "JS", "React",
"Redux"], ["Node", "Express", "MongoDB"]]
 console.log(fullStack.length)  // 2
 console.log(fullStack[0])  // ["HTML", "CSS", "JS", "React",
"Redux"]
 console.log(fullStack[1]) // ["Node", "Express", "MongoDB"]
```

You are diligent and you have already achieved quite a lot. You have just completed day 5 challenges and you are 5 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercise

Exercise: Level 1

```
const countries = [
  'Albania',
  'Bolivia',
  'Canada',
  'Denmark',
  'Ethiopia',
  'Finland',
  'Germany',
  'Hungary',
  'Ireland',
  'Japan',
  'Kenya'
]

const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
  'React',
  'Redux',
  'Node',
  'MongoDB'
]
```

1. Declare an *empty* array;
2. Declare an array with more than 5 number of elements
3. Find the length of your array
4. Get the first item, the middle item and the last item of the array
5. Declare an array called *mixedDataTypes*, put different data types in the array and find the length of the array. The array size should be greater than 5
6. Declare an array variable name itCompanies and assign initial values Facebook, Google, Microsoft, Apple, IBM, Oracle and Amazon
7. Print the array using *console.log()*
8. Print the number of companies in the array
9. Print the first company, middle and last company
10. Print out each company
11. Change each company name to uppercase one by one and print them out

12. Print the array like as a sentence: Facebook, Google, Microsoft, Apple, IBM,Oracle and Amazon are big IT companies.
13. Check if a certain company exists in the itCompanies array. If it exist return the company else return a company is *not found*
14. Filter out companies which have more than one 'o' without the filter method
15. Sort the array using *sort()* method
16. Reverse the array using *reverse()* method
17. Slice out the first 3 companies from the array
18. Slice out the last 3 companies from the array
19. Slice out the middle IT company or companies from the array
20. Remove the first IT company from the array
21. Remove the middle IT company or companies from the array
22. Remove the last IT company from the array
23. Remove all IT companies

Exercise: Level 2

1. Create a separate countries.js file and store the countries array in to this file, create a separate file web_techs.js and store the webTechs array in to this file. Access both file in main.js file
2. First remove all the punctuations and change the string to array and count the number of words in the array

```
let text =
'I love teaching and empowering people. I teach HTML,
CSS, JS, React, Python.'
console.log(words)
console.log(words.length)
```

```
["I", "love", "teaching", "and", "empowering", "people",
"I", "teach", "HTML", "CSS", "JS", "React", "Python"]

13
```

3. In the following shopping cart add, remove, edit items

```
const shoppingCart = ['Milk', 'Coffee', 'Tea', 'Honey']
```

    - add 'Meat' in the beginning of your shopping cart if it has not been already added
    - add Sugar at the end of you shopping cart if it has not been already added
    - remove 'Honey' if you are allergic to honey
    - modify Tea to 'Green Tea'
4. In countries array check if 'Ethiopia' exists in the array if it exists print 'ETHIOPIA'. If it does not exist add to the countries list.
5. In the webTechs array check if Sass exists in the array and if it exists print 'Sass is a CSS preprocess'. If it does not exist add Sass to the array and print the array.
6. Concatenate the following two variables and store it in a fullStack variable.

```
const frontEnd = ['HTML', 'CSS', 'JS', 'React', 'Redux']
const backEnd = ['Node','Express', 'MongoDB']

console.log(fullStack)
```

```
["HTML", "CSS", "JS", "React", "Redux", "Node", "Express",
"MongoDB"]
```

Exercise: Level 3

1. The following is an array of 10 students ages:

```
const ages = [19, 22, 19, 24, 20, 25, 26, 24, 25, 24]
```

    - Sort the array and find the min and max age
    - Find the median age(one middle item or two middle items divided by two)
    - Find the average age(all items divided by number of items)
    - Find the range of the ages(max minus min)
    - Compare the value of (min - average) and (max - average), use *abs()* method 1.Slice the first ten countries from the countries array

2. Find the middle country(ies) in the countries array
3. Divide the countries array into two equal arrays if it is even. If countries array is not even , one more country for the first half.

🎉 CONGRATULATIONS ! DAY-5 COMPLETED 🎉

DAY-6 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 impl–ement this simple task it may take you 2 to 5 minutes, such kind of tedious and repetitive task can be carried out using loop. If you prefer watching the videos, you can checkout the video tutorials

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

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

console.log(newArr)  // ["FINLAND", "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 = ['Finland', 'Sweden', 'Norway', 'Denmark',
'Iceland']
const newArr = []
for(const country of countries){
  newArr.push(country.toUpperCase())
}

console.log(newArr)  // ["FINLAND", "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
```

 You are so brave, you made it to this far. Now, you have gained the power to automate repetitive and tedious tasks. You have just completed day 6 challenges and you are 6 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises:Day 6

Exercises: Level 1

```
const countries = [
  'Albania',
  'Bolivia',
  'Canada',
  'Denmark',
  'Ethiopia',
  'Finland',
  'Germany',
  'Hungary',
  'Ireland',
  'Japan',
  'Kenya'
]

const webTechs = [
  'HTML',
  'CSS',
  'JavaScript',
  'React',
  'Redux',
  'Node',
  'MongoDB'
]

const mernStack = ['MongoDB', 'Express', 'React', 'Node']
```

1. Iterate 0 to 10 using for loop, do the same using while and do while loop
2. Iterate 10 to 0 using for loop, do the same using while and do while loop
3. Iterate 0 to n using for loop
4. Write a loop that makes the following pattern using console.log():

```
    #
    ##
    ###
    ####
    #####
    ######
    #######
```

5. Use loop to print the following pattern:

```
0 x 0 = 0
1 x 1 = 1
2 x 2 = 4
3 x 3 = 9
```

```
4 x 4 = 16
5 x 5 = 25
6 x 6 = 36
7 x 7 = 49
8 x 8 = 64
9 x 9 = 81
10 x 10 = 100
```

6. Using loop print the following pattern

```
 i    i^2   i^3
 0    0     0
 1    1     1
 2    4     8
 3    9     27
 4    16    64
 5    25    125
 6    36    216
 7    49    343
 8    64    512
 9    81    729
 10   100   1000
```

7. Use for loop to iterate from 0 to 100 and print only even numbers
8. Use for loop to iterate from 0 to 100 and print only odd numbers
9. Use for loop to iterate from 0 to 100 and print only prime numbers
10. Use for loop to iterate from 0 to 100 and print the sum of all numbers.

```
The sum of all numbers from 0 to 100 is 5050.
```

11. Use for loop to iterate from 0 to 100 and print the sum of all evens and the sum of all odds.

```
The sum of all evens from 0 to 100 is 2550. And the sum
of all odds from 0 to 100 is 2500.
```

12. Use for loop to iterate from 0 to 100 and print the sum of all evens and the sum of all odds. Print sum of evens and sum of odds as array

```
  [2550, 2500]
```

13. Develop a small script which generate array of 5 random numbers
14. Develop a small script which generate array of 5 random numbers and the numbers must be unique
15. Develop a small script which generate a six characters random id:

```
5j2khz
```

Exercises: Level 2

1. Develop a small script which generate any number of characters random id:

```
  fe3jo1gl124g
  xkqci4utda1lmbelpkm03rba
```

2. Write a script which generates a random hexadecimal number.

```
'#ee33df'
```

3. Write a script which generates a random rgb color number.

```
rgb(240,180,80)
```

4. Using the above countries array, create the following new array.

```
["ALBANIA", "BOLIVIA", "CANADA", "DENMARK", "ETHIOPIA",
"FINLAND", "GERMANY", "HUNGARY", "IRELAND", "JAPAN",
"KENYA"]
```

5. Using the above countries array, create an array for countries length'.

```
[7, 7, 6, 7, 8, 7, 7, 7, 7, 5, 5]
```

6. Use the countries array to create the following array of arrays:

```
  [
  ['Albania', 'ALB', 7],
  ['Bolivia', 'BOL', 7],
  ['Canada', 'CAN', 6],
  ['Denmark', 'DEN', 7],
  ['Ethiopia', 'ETH', 8],
  ['Finland', 'FIN', 7],
  ['Germany', 'GER', 7],
  ['Hungary', 'HUN', 7],
  ['Ireland', 'IRE', 7],
  ['Iceland', 'ICE', 7],
  ['Japan', 'JAP', 5],
  ['Kenya', 'KEN', 5]
]
```

7. In above countries array, check if there is a country or countries containing the word 'land'. If there are countries containing 'land', print it as array. If there is no country containing the word 'land', print 'All these countries are without land'.

```
['Finland','Ireland', 'Iceland']
```

8. In above countries array, check if there is a country or countries end with a substring 'ia'. If there are countries end with, print it as array. If there is no country containing the word 'ai', print 'These are countries ends without ia'.

```
['Albania', 'Bolivia','Ethiopia']
```

9. Using the above countries array, find the country containing the biggest number of characters.

```
Ethiopia
```

10. Using the above countries array, find the country containing only 5 characters.

```
['Japan', 'Kenya']
```

11. Find the longest word in the webTechs array
12. Use the webTechs array to create the following array of arrays:

```
[["HTML", 4], ["CSS", 3],["JavaScript", 10],["React",
5],["Redux", 5],["Node", 4],["MongoDB", 7]]
```

13. An application created using MongoDB, Express, React and Node is called a MERN stack app. Create the acronym MERN by using the array mernStack
14. Iterate through the array, ["HTML", "CSS", "JS", "React", "Redux", "Node", "Express", "MongoDB"] using a for loop or for of loop and print out the items.
15. This is a fruit array , ['banana', 'orange', 'mango', 'lemon'] reverse the order using loop without using a reverse method.
16. Print all the elements of array as shown below.

```
const fullStack = [
  ['HTML', 'CSS', 'JS', 'React'],
  ['Node', 'Express', 'MongoDB']
]
```

```
HTML
CSS
JS
REACT
NODE
EXPRESS
MONGODB
```

Exercises: Level 3

1. Copy countries array(Avoid mutation)
2. Arrays are mutable. Create a copy of array which does not modify the original. Sort the copied array and store in a variable sortedCountries
3. Sort the webTechs array and mernStack array
4. Extract all the countries contain the word 'land' from the countries array and print it as array
5. Find the country containing the hightest number of characters in the countries array
6. Extract all the countries contain the word 'land' from the countries array and print it as array
7. Extract all the countries containing only four characters from the countries array and print it as array
8. Extract all the countries containing two or more words from the countries array and print it as array
9. Reverse the countries array and capitalize each country and stored it as an array

🎉 CONGRATULATIONS ! 🎉DAY-6 COMPELETED

DAY-7 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*

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 = 'Asabeneh'
      let lastName = 'Yetayeh'
      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 = 'Asabeneh'
      let lastName = 'Yetayeh'
      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('Asabeneh', 'Yetayeh'))
```

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))
```

Function with unlimited number of parameters

Sometimes we do not know how many arguments the user going to pass. Therefore, we should know how to write a function which can take unlimited number of arguments. The way we do it has a significant difference between a function declaration(regular function) and arrow function. Let us see examples both in function declaration and arrow function.

Unlimited number of parameters in regular function

A function declaration provides a function scoped arguments array like object. Any thing we passed as argument in the function can be accessed from arguments object inside the functions. Let us see an example

```
// Let us access the arguments object

function sumAllNums() {
 console.log(arguments)
}

sumAllNums(1, 2, 3, 4)
// Arguments(4) [1, 2, 3, 4, callee: ƒ,
Symbol(Symbol.iterator): ƒ]
```

```
// function declaration

function sumAllNums() {
  let sum = 0
  for (let i = 0; i < arguments.length; i++) {
```

```
    sum += arguments[i]
  }
  return sum
}

console.log(sumAllNums(1, 2, 3, 4)) // 10
console.log(sumAllNums(10, 20, 13, 40, 10))  // 93
console.log(sumAllNums(15, 20, 30, 25, 10, 33, 40))  // 173
```

Unlimited number of parameters in arrow function

Arrow function does not have the function scoped arguments object. To implement a function which takes unlimited number of arguments in an arrow function we use spread operator followed by any parameter name. Any thing we passed as argument in the function can be accessed as array in the arrow function. Let us see an example

```
// Let us access the arguments object

const sumAllNums = (...args) => {
 // console.log(arguments), arguments object not found in
arrow function
 // instead we use a parameter followed by spread operator
(...)
 console.log(args)
}

sumAllNums(1, 2, 3, 4)
// [1, 2, 3, 4]
```

```
// function declaration

const sumAllNums = (...args) => {
  let sum = 0
  for (const element of args) {
    sum += element
  }
  return sum
}

console.log(sumAllNums(1, 2, 3, 4)) // 10
console.log(sumAllNums(10, 20, 13, 40, 10))  // 93
console.log(sumAllNums(15, 20, 30, 25, 10, 33, 40))  // 173
```

Anonymous Function

Anonymous function or without name

```
const anonymousFun = function() {
  console.log(
    'I am an anonymous function and my value is stored in
anonymousFun'
  )
}
```

Expression Function

Expression functions are anonymous functions. After we create a function without a name and we assign it to a variable. To return a value from the function we should call the variable. Look at the example below.

```
// Function expression
const square = function(n) {
  return n * n
}

console.log(square(2)) // -> 4
```

Self Invoking Functions

Self invoking functions are anonymous functions which do not need to be called to return a value.

```
(function(n) {
  console.log(n * n)
})(2) // 4, but instead of just printing if we want to return
and store the data, we do as shown below

let squaredNum = (function(n) {
  return n * n
})(10)

console.log(squaredNum)
```

Arrow Function

Arrow function is an alternative to write a function, however function declaration and arrow function have some minor differences.

Arrow function uses arrow instead of the keyword *function* to declare a function. Let us see both function declaration and arrow function.

```
// This is how we write normal or declaration function
// Let us change this declaration function to an arrow
function
function square(n) {
  return n * n
}

console.log(square(2)) // 4

const square = n => {
  return n * n
}

console.log(square(2))  // -> 4

// if we have only one line in the code block, it can be
written as follows, explicit return
const square = n => n * n  // -> 4
```

```
const changeToUpperCase = arr => {
  const newArr = []
  for (const element of arr) {
    newArr.push(element.toUpperCase())
  }
  return newArr
}

const countries = ['Finland', 'Sweden', 'Norway', 'Denmark',
'Iceland']
console.log(changeToUpperCase(countries))

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

```
const printFullName = (firstName, lastName) => {
  return `${firstName} ${lastName}`
}

console.log(printFullName('Asabeneh', 'Yetayeh'))
```

The above function has only the return statement, therefore, we can explicitly return it as follows.

```
const printFullName = (firstName, lastName) => `${firstName} ${lastName}`

console.log(printFullName('Asabeneh', 'Yetayeh'))
```

Function with default parameters

Sometimes we pass default values to parameters, when we invoke the function if we do not pass an argument the default value will be used. Both function declaration and arrow function can have a default value or values.

```
// syntax
// Declaring a function
function functionName(param = value) {
  //codes
}

// Calling function
functionName()
functionName(arg)
```

Example:

```
function greetings(name = 'Peter') {
  let message = `${name}, welcome to 30 Days Of JavaScript!`
  return message
}

console.log(greetings())
console.log(greetings('Asabeneh'))
```

```
function generateFullName(firstName = 'Asabeneh', lastName =
'Yetayeh') {
  let space = ' '
  let fullName = firstName + space + lastName
  return fullName
}

console.log(generateFullName())
console.log(generateFullName('David', 'Smith'))
```

```
function calculateAge(birthYear, currentYear = 2019) {
  let age = currentYear - birthYear
  return age
}

console.log('Age: ', calculateAge(1819))
```

```
function weightOfObject(mass, gravity = 9.81) {
  let weight = mass * gravity + ' N' // the value has to be
changed to string first
  return weight
}

console.log('Weight of an object in Newton: ',
weightOfObject(100)) // 9.81 gravity at the surface of Earth
console.log('Weight of an object in Newton: ',
weightOfObject(100, 1.62)) // gravity at surface of Moon
```

Let us see how we write the above functions with arrow functions

```
// syntax
// Declaring a function
const functionName = (param = value) => {
  //codes
}

// Calling function
functionName()
functionName(arg)
```

Example:

```
const greetings = (name = 'Peter') => {
  let message = name + ', welcome to 30 Days Of JavaScript!'
  return message
}

console.log(greetings())
console.log(greetings('Asabeneh'))
```

```
const generateFullName = (firstName = 'Asabeneh', lastName =
'Yetayeh') => {
  let space = ' '
  let fullName = firstName + space + lastName
  return fullName
}

console.log(generateFullName())
console.log(generateFullName('David', 'Smith'))
```

```
const calculateAge = (birthYear, currentYear = 2019) =>
currentYear - birthYear
console.log('Age: ', calculateAge(1819))
```

```
const weightOfObject = (mass, gravity = 9.81) => mass *
gravity + ' N'

console.log('Weight of an object in Newton: ',
weightOfObject(100)) // 9.81 gravity at the surface of Earth
console.log('Weight of an object in Newton: ',
weightOfObject(100, 1.62)) // gravity at surface of Moon
```

Function declaration versus Arrow function

It Will be covered in other section.

🥏 You are a rising star, now you knew function . Now, you are super charged with the power of functions. You have just completed day 7 challenges and you are 7 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises

Exercises: Level 1

1. Declare a function *fullName* and it print out your full name.
2. Declare a function *fullName* and now it takes firstName, lastName as a parameter and it returns your full - name.
3. Declare a function *addNumbers* and it takes two two parameters and it returns sum.
4. An area of a rectangle is calculated as follows: *area = length x width*. Write a function which calculates *areaOfRectangle*.
5. A perimeter of a rectangle is calculated as follows: *perimeter= 2x(length + width)*. Write a function which calculates *perimeterOfRectangle*.
6. A volume of a rectangular prism is calculated as follows: *volume = length x width x height*. Write a function which calculates *volumeOfRectPrism*.
7. Area of a circle is calculated as follows: *area = π x r x r*. Write a function which calculates *areaOfCircle*
8. Circumference of a circle is calculated as follows: *circumference = 2πr*. Write a function which calculates *circumOfCircle*
9. Density of a substance is calculated as follows:*density= mass/volume*. Write a function which calculates *density*.
10. Speed is calculated by dividing the total distance covered by a moving object divided by the total amount of time taken. Write a function which calculates a speed of a moving object, *speed*.
11. Weight of a substance is calculated as follows: *weight = mass x gravity*. Write a function which calculates *weight*.
12. Temperature in oC can be converted to oF using this formula: *oF = (oC x 9/5) + 32*. Write a function which convert oC to oF *convertCelsiusToFahrenheit*.
13. Body mass index(BMI) is calculated as follows: *bmi = weight in Kg / (height x height) in m2*. Write a function which calculates *bmi*. BMI is used to broadly define different weight groups in adults 20 years old or older.Check if a

person is *underweight, normal, overweight* or *obese* based the information given below.

- The same groups apply to both men and women.
- *Underweight*: BMI is less than 18.5
- *Normal weight*: BMI is 18.5 to 24.9
- *Overweight*: BMI is 25 to 29.9
- *Obese*: BMI is 30 or more

14. Write a function called *checkSeason*, it takes a month parameter and returns the season:Autumn, Winter, Spring or Summer.
15. Math.max returns its largest argument. Write a function findMax that takes three arguments and returns their maximum with out using Math.max method.

```
console.log(findMax(0, 10, 5))
10
console.log(findMax(0, -10, -2))
0
```

Exercises: Level 2

1. Linear equation is calculated as follows: *ax + by + c = 0*. Write a function which calculates value of a linear equation, *solveLinEquation*.
2. Quadratic equation is calculated as follows: *ax2 + bx + c = 0*. Write a function which calculates value or values of a quadratic equation, *solveQuadEquation*.

```
console.log(solveQuadratic()) // {0}
console.log(solveQuadratic(1, 4, 4)) // {-2}
console.log(solveQuadratic(1, -1, -2)) // {2, -1}
console.log(solveQuadratic(1, 7, 12)) // {-3, -4}
console.log(solveQuadratic(1, 0, -4)) //{2, -2}
console.log(solveQuadratic(1, -1, 0)) //{1, 0}
```

3. Declare a function name *printArray*. It takes array as a parameter and it prints out each value of the array.
4. Write a function name *showDateTime* which shows time in this format: 08/01/2020 04:08 using the Date object.

```
showDateTime()
08/01/2020 04:08
```

5. Declare a function name *swapValues*. This function swaps value of x to y.

```
swapValues(3, 4) // x => 4, y=>3
swapValues(4, 5) // x = 5, y = 4
```

6. Declare a function name *reverseArray*. It takes array as a parameter and it returns the reverse of the array (don't use method).

```
console.log(reverseArray([1, 2, 3, 4, 5]))
//[5, 4, 3, 2, 1]
console.log(reverseArray(['A', 'B', 'C']))
//['C', 'B', 'A']
```

7. Declare a function name *capitalizeArray*. It takes array as a parameter and it returns the - capitalizedarray.
8. Declare a function name *addItem*. It takes an item parameter and it returns an array after adding the item
9. Declare a function name *removeItem*. It takes an index parameter and it returns an array after removing an item
10. Declare a function name *sumOfNumbers*. It takes a number parameter and it adds all the numbers in that range.
11. Declare a function name *sumOfOdds*. It takes a number parameter and it adds all the odd numbers in that - range.
12. Declare a function name *sumOfEven*. It takes a number parameter and it adds all the even numbers in that - range.
13. Declare a function name evensAndOdds . It takes a positive integer as parameter and it counts number of evens and odds in the number.

```
evensAndOdds(100);
The number of odds are 50.
The number of evens are 51.
```

14. Write a function which takes any number of arguments and return the sum of the arguments

```
sum(1, 2, 3) // -> 6
sum(1, 2, 3, 4) // -> 10
```

15. Writ a function which generates a *randomUserIp*.
16. Write a function which generates a *randomMacAddress*

17. Declare a function name *randomHexaNumberGenerator*. When this function is called it generates a random hexadecimal number. The function return the hexadecimal number.

```
console.log(randomHexaNumberGenerator());
'#ee33df'
```

18. Declare a function name *userIdGenerator*. When this function is called it generates seven character id. The function return the id.

```
console.log(userIdGenerator());
41XTDbE
```

Exercises: Level 3

1. Modify the *userIdGenerator* function. Declare a function name *userIdGeneratedByUser*. It doesn't take any parameter but it takes two inputs using prompt(). One of the input is the number of characters and the second input is the number of ids which are supposed to be generated.

```
userIdGeneratedByUser()
'kcsy2
SMFYb
bWmeq
ZXOYh
2Rgxf
'
userIdGeneratedByUser()
'1GCSgPLMaBAVQZ26
YD7eFwNQKNs7qXaT
ycArC5yrRupyG00S
UbGxOFI7UXSWAyKN
dIV0SSUTgAdKwStr
'
```

2. Write a function name *rgbColorGenerator* and it generates rgb colors.

```
rgbColorGenerator()
rgb(125,244,255)
```

3. Write a function ***arrayOfHexaColors*** which return any number of hexadecimal colors in an array.
4. Write a function ***arrayOfRgbColors*** which return any number of RGB colors in an array.

5. Write a function ***convertHexaToRgb*** which converts hexa color to rgb and it returns an rgb color.
6. Write a function ***convertRgbToHexa*** which converts rgb to hexa color and it returns an hexa color.
7. Write a function ***generateColors*** which can generate any number of hexa or rgb colors.

```
console.log(generateColors('hexa', 3)) // ['#a3e12f',
'#03ed55', '#eb3d2b']
console.log(generateColors('hexa', 1)) // '#b334ef'
console.log(generateColors('rgb', 3)) // ['rgb(5, 55,
175)', 'rgb(50, 105, 100)', 'rgb(15, 26, 80)']
console.log(generateColors('rgb', 1)) // 'rgb(33,79,
176)'
```

8. Call your function *shuffleArray*, it takes an array as a parameter and it returns a shuffled array
9. Call your function *factorial*, it takes a whole number as a parameter and it return a factorial of the number
10. Call your function *isEmpty*, it takes a parameter and it checks if it is empty or not
11. Call your function *sum*, it takes any number of arguments and it returns the sum.
12. Write a function called *sumOfArrayItems*, it takes an array parameter and return the sum of all the items. Check if all the array items are number types. If not give return reasonable feedback.
13. Write a function called *average*, it takes an array parameter and returns the average of the items. Check if all the array items are number types. If not give return reasonable feedback.
14. Write a function called *modifyArray* takes array as parameter and modifies the fifth item of the array and return the array. If the array length is less than five it return 'item not found'.

```
console.log(modifyArray(['Avocado', 'Tomato',
'Potato','Mango', 'Lemon','Carrot']);

['Avocado', 'Tomato', 'Potato','Mango', 'LEMON',
'Carrot']
```

```
console.log(modifyArray(['Google', 'Facebook','Apple', 'Amazon','Microsoft',  'IBM']);

['Google', 'Facebook','Apple', 'Amazon','MICROSOFT', 'IBM']

console.log(modifyArray(['Google', 'Facebook','Apple', 'Amazon']);

  'Not Found'
```

15. Write a function called *isPrime*, which checks if a number is prime number.
16. Write a functions which checks if all items are unique in the array.
17. Write a function which checks if all the items of the array are the same data type.
18. JavaScript variable name does not support special characters or symbols except $ or _. Write a function **isValidVariable** which check if a variable is valid or invalid variable.
19. Write a function which returns array of seven random numbers in a range of 0-9. All the numbers must be unique.

```
sevenRandomNumbers()
[(1, 4, 5, 7, 9, 8, 0)]
```

20. Write a function called reverseCountries, it takes countries array and first it copy the array and returns the reverse of the original array

🎉 CONGRATULATIONS ! DAY-7 COMPLETED 🎉

DAY-8 OBJECTS

Scope

Variable is the fundamental part in programming. We declare variable to store different data types. To declare a variable we use the key word *var*, *let* and *const*. A variable can be declared at different scope. In this section, we will see the scope variables, scope of variables when we use var or let. Variables scopes can be:

- Global
- Local

Variable can be declared globally or locally scope. We will see both global and local scope. Anything declared without let, var or const is scoped at global level.

Let us imagine that we have a scope.js file.

Window Global Object

Without using console.log() open your browser and check, you will see the value of a and b if you write a or b on the browser. That means a and b are already available in the window.

```
//scope.js
a = 'JavaScript' // declaring a variable without let or const
make it available in window object and this found anywhere
b = 10 // this is a global scope variable and found in the
window object
function letsLearnScope() {
  console.log(a, b)
  if (true) {
    console.log(a, b)
  }
}
console.log(a, b) // accessible
```

Global scope

A globally declared variable can be accessed every where in the same file. But the term global is relative. It can be global to the file or it can be global relative to some block of codes.

```
//scope.js
let a = 'JavaScript' // is a global scope it will be found
anywhere in this file
let b = 10 // is a global scope it will be found anywhere in
this file
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  if (true) {
    let a = 'Python'
    let b = 100
    console.log(a, b) // Python 100
  }
  console.log(a, b)
}
letsLearnScope()
console.log(a, b) // JavaScript 10, accessible
```

Local scope

A variable declared as local can be accessed only in certain block code.

- Block Scope
- Function Scope

```
//scope.js
let a = 'JavaScript' // is a global scope it will be found
anywhere in this file
let b = 10 // is a global scope it will be found anywhere in
this file
// Function scope
function letsLearnScope() {
  console.log(a, b) // JavaScript 10, accessible
  let value = false
// block scope
  if (true) {
    // we can access from the function and outside the
function but
```

```
    // variables declared inside the if will not be accessed
outside the if block
    let a = 'Python'
    let b = 20
    let c = 30
    let d = 40
    value = !value
    console.log(a, b, c, value) // Python 20 30 true
  }
  // we can not access c because c's scope is only the if
block
  console.log(a, b, value) // JavaScript 10 true
}
letsLearnScope()
console.log(a, b) // JavaScript 10, accessible
```

Now, you have an understanding of scope. A variable declared with *var* only scoped to function but variable declared with *let* or *const* is block scope(function block, if block, loop block, etc). Block in JavaScript is a code in between two curly brackets ({}).

```
//scope.js
function letsLearnScope() {
  var gravity = 9.81
  console.log(gravity)

}
// console.log(gravity), Uncaught ReferenceError: gravity is
not defined

if (true){
  var gravity = 9.81
  console.log(gravity) // 9.81
}
console.log(gravity)  // 9.81

for(var i = 0; i < 3; i++){
  console.log(i) // 0, 1, 2
}
console.log(i) // 3
```

In ES6 and above there is *let* and *const*, so you will not suffer from the sneakiness of *var*. When we use *let* our variable is block scoped and it will not infect other parts of our code.

```
//scope.js
```

```
function letsLearnScope() {
  // you can use let or const, but gravity is constant I
prefer to use const
  const gravity = 9.81
  console.log(gravity)

}
// console.log(gravity), Uncaught ReferenceError: gravity is
not defined

if (true){
  const  gravity = 9.81
  console.log(gravity) // 9.81
}
// console.log(gravity), Uncaught ReferenceError: gravity is
not defined

for(let i = 0; i < 3; i++){
  console.log(i) // 0, 1, 2
}
// console.log(i), Uncaught ReferenceError: i is not defined
```

The scope *let* and *const* are the same. The difference is only reassigning. We can not change or reassign the value of the const variable. I would strongly suggest you to use *let* and *const*, by using *let* and *const* you will write clean code and avoid hard to debug mistakes. As a rule of thumb, you can use *let* for any value which change, *const* for any constant value, and for an array, object, arrow function and function expression.

Object

Everything can be an object and objects do have properties and properties have values, so an object is a key value pair. The order of the key is not reserved, or there is no order. To create an object literal, we use two curly brackets.

Creating an empty object

An empty object

```
const person = {}
```

Creating an objecting with values

Now, the person object has firstName, lastName, age, location, skills and isMarried properties. The value of properties or keys could be a string, number, boolean, an object, null, undefined or a function.

Let us see some examples of object. Each key has a value in the object.

```
const rectangle = {
  length: 20,
  width: 20
}
console.log(rectangle) // {length: 20, width: 20}

const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  isMarried: true
}
console.log(person)
```

Getting values from an object

We can access values of object using two methods:

- using . followed by key name if the key-name is a one word
- using square bracket and a quote

```
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
```

```
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName}${this.lastName}`
  },
  'phone number': '+3584545454545'
}

// accessing values using .
console.log(person.firstName)
console.log(person.lastName)
console.log(person.age)
console.log(person.location) // undefined

// value can be accessed using square bracket and key name
console.log(person['firstName'])
console.log(person['lastName'])
console.log(person['age'])
console.log(person['age'])
console.log(person['location']) // undefined

// for instance to access the phone number we only use the
square bracket method
console.log(person['phone number'])
```

Creating object methods

Now, the person object has getFullName properties. The getFullName is function inside the person object and we call it an object method. The *this* key word refers to the object itself. We can use the word *this* to access the values of different properties of the object. We can not use an arrow function as object method because the word this refers to the window inside an arrow function instead of the object itself. Example of object:

```
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
```

```
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName} ${this.lastName}`
  }
}

console.log(person.getFullName())
// Asabeneh Yetayeh
```

Setting new key for an object

An object is a mutable data structure and we can modify the content of an object after it gets created.

Setting a new keys in an object

```
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  getFullName: function() {
    return `${this.firstName} ${this.lastName}`
  }
}
person.nationality = 'Ethiopian'
person.country = 'Finland'
person.title = 'teacher'
```

```
person.skills.push('Meteor')
person.skills.push('SasS')
person.isMarried = true

person.getPersonInfo = function() {
  let skillsWithoutLastSkill = this.skills
    .splice(0, this.skills.length - 1)
    .join(', ')
  let lastSkill = this.skills.splice(this.skills.length -
1)[0]

  let skills = `${skillsWithoutLastSkill}, and ${lastSkill}`
  let fullName = this.getFullName()
  let statement = `${fullName} is a ${this.title}.\nHe lives
in ${this.country}.\nHe teaches ${skills}.`
  return statement
}
console.log(person)
console.log(person.getPersonInfo())
```

```
Asabeneh Yetayeh is a teacher.
He lives in Finland.
He teaches HTML, CSS, JavaScript, React, Node, MongoDB,
Python, D3.js, Meteor, and SasS.
```

Object Methods

There are different methods to manipulate an object. Let us see some of the available methods.

Object.assign: To copy an object without modifying the original object

```
const person = {
  firstName: 'Asabeneh',
  age: 250,
  country: 'Finland',
  city:'Helsinki',
  skills: ['HTML', 'CSS', 'JS'],
  title: 'teacher',
  address: {
    street: 'Heitamienkatu 16',
    pobox: 2002,
    city: 'Helsinki'
  },
  getPersonInfo: function() {
    return `I am ${this.firstName} and I live in ${this.city},
${this.country}. I am ${this.age}.`
```

```
  }
}

//Object methods: Object.assign, Object.keys, Object.values,
Object.entries
//hasOwnProperty

const copyPerson = Object.assign({}, person)
console.log(copyPerson)
```

Getting object keys using Object.keys()

Object.keys: To get the keys or properties of an object as an array

```
const keys = Object.keys(copyPerson)
console.log(keys) //['firstName', 'age', 'country','city',
'skills','title', 'address', 'getPersonInfo']
const address = Object.keys(copyPerson.address)
console.log(address) //['street', 'pobox', 'city']
```

Getting object values using Object.values()

Object.values:To get values of an object as an array

```
const values = Object.values(copyPerson)
console.log(values)
```

Getting object keys and values using Object.entries()

Object.entries:To get the keys and values in an array

```
const entries = Object.entries(copyPerson)
console.log(entries)
```

Checking properties using hasOwnProperty()

hasOwnProperty: To check if a specific key or property exist in an object

```
console.log(copyPerson.hasOwnProperty('name'))
console.log(copyPerson.hasOwnProperty('score'))
```

You are astonishing. Now, you are super charged with the power of objects. You have just completed day 8 challenges and you are 8 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises

Exercises: Level 1

1. Create an empty object called dog
2. Print the the dog object on the console
3. Add name, legs, color, age and bark properties for the dog object. The bark property is a method which return *woof woof*
4. Get name, legs, color, age and bark value from the dog object
5. Set new properties the dog object: breed, getDogInfo

Exercises: Level 2

1. Find the person who has many skills in the users object.
2. Count logged in users, count users having greater than equal to 50 points from the following object.

```
const users = {
  Alex: {
    email: 'alex@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript'],
    age: 20,
    isLoggedIn: false,
    points: 30
  },
  Asab: {
    email: 'asab@asab.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'Redux', 'MongoDB',
'Express', 'React', 'Node'],
    age: 25,
    isLoggedIn: false,
    points: 50
  },
  Brook: {
    email: 'daniel@daniel.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux'],
    age: 30,
    isLoggedIn: true,
    points: 50
  },
  Daniel: {
    email: 'daniel@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'Python'],
```

```
    age: 20,
    isLoggedIn: false,
    points: 40
  },
  John: {
    email: 'john@john.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux',
'Node.js'],
    age: 20,
    isLoggedIn: true,
    points: 50
  },
  Thomas: {
    email: 'thomas@thomas.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React'],
    age: 20,
    isLoggedIn: false,
    points: 40
  },
  Paul: {
    email: 'paul@paul.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'MongoDB',
'Express', 'React', 'Node'],
    age: 20,
    isLoggedIn: false,
    points: 40
  }
}```
```

3. Find people who are MERN stack developer from the users object
4. Set your name in the users object without modifying the original users object
5. Get all keys or properties of users object
6. Get all the values of users object
7. Use the countries object to print a country name, capital, populations and languages.

Exercises: Level 3

1. Create an object literal called *personAccount*. It has *firstName, lastName, incomes, expenses* properties and it has *totalIncome, totalExpense, accountInfo,addIncome, addExpense* and *accountBalance* methods. Incomes is a set of incomes and its description and expenses is a set of incomes and its description.
2. **** Questions:2, 3 and 4 are based on the following two arrays:users and products ()

```
const users = [
{
    _id: 'ab12ex',
    username: 'Alex',
    email: 'alex@alex.com',
    password: '123123',
    createdAt:'08/01/2020 9:00 AM',
    isLoggedIn: false
},
{
    _id: 'fg12cy',
    username: 'Asab',
    email: 'asab@asab.com',
    password: '123456',
    createdAt:'08/01/2020 9:30 AM',
    isLoggedIn: true
},
{
    _id: 'zwf8md',
    username: 'Brook',
    email: 'brook@brook.com',
    password: '123111',
    createdAt:'08/01/2020 9:45 AM',
    isLoggedIn: true
},
{
    _id: 'eefamr',
    username: 'Martha',
    email: 'martha@martha.com',
    password: '123222',
    createdAt:'08/01/2020 9:50 AM',
    isLoggedIn: false
},
{
    _id: 'ghderc',
    username: 'Thomas',
    email: 'thomas@thomas.com',
    password: '123333',
    createdAt:'08/01/2020 10:00 AM',
    isLoggedIn: false
```

```
    }
    ];

    const products = [
  {
    _id: 'eedfcf',
    name: 'mobile phone',
    description: 'Huawei Honor',
    price: 200,
    ratings: [
      { userId: 'fg12cy', rate: 5 },
      { userId: 'zwf8md', rate: 4.5 }
    ],
    likes: []
  },
  {
    _id: 'aegfal',
    name: 'Laptop',
    description: 'MacPro: System Darwin',
    price: 2500,
    ratings: [],
    likes: ['fg12cy']
  },
  {
    _id: 'hedfcg',
    name: 'TV',
    description: 'Smart TV:Procaster',
    price: 400,
    ratings: [{ userId: 'fg12cy', rate: 5 }],
    likes: ['fg12cy']
  }
]
```

Imagine you are getting the above users collection from a MongoDB database. a. Create a function called signUp which allows user to add to the collection. If user exists, inform the user that he has already an account.
b. Create a function called signIn which allows user to sign in to the application

3. The products array has three elements and each of them has six properties. a. Create a function called rateProduct which rates the product b. Create a function called averageRating which calculate the average rating of a product
4. Create a function called likeProduct. This function will helps to like to the product if it is not liked and remove like if it was liked.

🎉 CONGRATULATIONS ! DAY-8 COMPLETED 🎉

DAY-9 HIGHER ORDER FUNCTIONS

Higher order functions are functions which take other function as a parameter or return a function as a value. The function passed as a parameter is called callback.

Callback

A callback is a function which can be passed as parameter to other function. See the example below.

```
// a callback function, the name of the function could be any
name
const callback = (n) => {
  return n ** 2
}

// function that takes other function as a callback
function cube(callback, n) {
  return callback(n) * n
}

console.log(cube(callback, 3))
```

Returning function

Higher order functions return function as a value

```
// Higher order function returning an other function
const higherOrder = n => {
  const doSomething = m => {
    const doWhatEver = t => {
      return 2 * n + 3 * m + t
    }
    return doWhatEver
  }
  return doSomething
}
console.log(higherOrder(2)(3)(10))
```

Let us see were we use call back functions. For instance the *forEach* method uses call back.

```
const numbers = [1, 2, 3, 4, 5]
const sumArray = arr => {
  let sum = 0
  const callback = function(element) {
    sum += element
  }
  arr.forEach(callback)
  return sum

}
console.log(sumArray(numbers))
```

```
15
```

The above example can be simplified as follows:

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

const sumArray = arr => {
  let sum = 0
  arr.forEach(function(element) {
    sum += element
  })
  return sum

}
console.log(sumArray(numbers))
```

```
15
```

Setting time

In JavaScript we can execute some activities in a certain interval of time or we can schedule(wait) for some time to execute some activities.

- setInterval
- setTimeout

Setting Interval using a setInterval function

In JavaScript, we use setInterval higher order function to do some activity continuously with in some interval of time. The setInterval global method take a callback function and a duration as a parameter. The duration is in milliseconds and the callback will be always called in that interval of time.

```
// syntax
function callback() {
  // code goes here
}
setInterval(callback, duration)
```

```
function sayHello() {
  console.log('Hello')
}
setInterval(sayHello, 1000) // it prints hello in every
second, 1000ms is 1s
```

Setting a time using a setTimeout

In JavaScript, we use setTimeout higher order function to execute some action at some time in the future. The setTimeout global method take a callback function and a duration as a parameter. The duration is in milliseconds and the callback wait for that amount of time.

```
// syntax
function callback() {
  // code goes here
}
setTimeout(callback, duration) // duration in milliseconds
```

```
function sayHello() {
  console.log('Hello')
}
setTimeout(sayHello, 2000) // it prints hello after it waits
for 2 seconds.
```

Functional Programming

Instead of writing regular loop, latest version of JavaScript introduced lots of built in methods which can help us to solve complicated problems. All builtin methods take callback function. In this section, we will
see *forEach*, *map*, *filter*, *reduce*, *find*, *every*, *some*, and *sort*.

forEach

forEach: Iterate an array elements. We use *forEach* only with arrays. It takes a callback function with elements, index parameter and array itself. The index and the array optional.

```
arr.forEach(function (element, index, arr) {
  console.log(index, element, arr)
})
// The above code can be written using arrow function
arr.forEach((element, index, arr) => {
  console.log(index, element, arr)
})
// The above code can be written using arrow function and
explicit return
arr.forEach((element, index, arr) => console.log(index,
element, arr))
```

```
let sum = 0;
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(num => console.log(num))
console.log(sum)
```

```
1
2
3
4
5
```

```
let sum = 0;
const numbers = [1, 2, 3, 4, 5];
numbers.forEach(num => sum += num)

console.log(sum)
```

```
15
```

```
const countries = ['Finland', 'Denmark', 'Sweden', 'Norway',
'Iceland']
countries.forEach((element) =>
console.log(element.toUpperCase()))
```

```
FINLAND
DENMARK
SWEDEN
NORWAY
ICELAND
```

map

map: Iterate an array elements and modify the array elements. It takes a callback function with elements, index , array parameter and return a new array.

```
const modifiedArray = arr.map(function (element, index, arr) {
  return element
})
```

```
/*Arrow function and explicit return
const modifiedArray = arr.map((element,index) => element);
*/
//Example
const numbers = [1, 2, 3, 4, 5]
const numbersSquare = numbers.map((num) => num * num)

console.log(numbersSquare)
```

```
[1, 4, 9, 16, 25]
```

```
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const namesToUpperCase = names.map((name) =>
name.toUpperCase())
console.log(namesToUpperCase)
```

```
['ASABENEH', 'MATHIAS', 'ELIAS', 'BROOK']
```

```
const countries = [
  'Albania',
  'Bolivia',
  'Canada',
  'Denmark',
  'Ethiopia',
  'Finland',
  'Germany',
  'Hungary',
  'Ireland',
  'Japan',
  'Kenya',
]
const countriesToUpperCase = countries.map((country) =>
country.toUpperCase())
console.log(countriesToUpperCase)

/*
// Arrow function
const countriesToUpperCase = countries.map((country) => {
  return country.toUpperCase();
})
//Explicit return arrow function
const countriesToUpperCase = countries.map(country =>
country.toUpperCase());
*/
```

```
['ALBANIA', 'BOLIVIA', 'CANADA', 'DENMARK', 'ETHIOPIA',
'FINLAND', 'GERMANY', 'HUNGARY', 'IRELAND', 'JAPAN', 'KENYA']
```

```
const countriesFirstThreeLetters = countries.map((country) =>
  country.toUpperCase().slice(0, 3)
)
```

```
 ["ALB", "BOL", "CAN", "DEN", "ETH", "FIN", "GER", "HUN",
"IRE", "JAP", "KEN"]
```

filter

Filter: Filter out items which full fill filtering conditions and return a new array.

```
//Filter countries containing land
const countriesContainingLand = countries.filter((country) =>
  country.includes('land')
)
console.log(countriesContainingLand)
```

```
['Finland', 'Ireland']
```

```
const countriesEndsByia = countries.filter((country) =>
country.endsWith('ia'))
console.log(countriesEndsByia)
```

```
['Albania', 'Bolivia','Ethiopia']
```

```
const countriesHaveFiveLetters = countries.filter(
  (country) => country.length === 5
)
console.log(countriesHaveFiveLetters)
```

```
['Japan', 'Kenya']
```

```
const scores = [
  { name: 'Asabeneh', score: 95 },
   { name: 'Lidiya', score: 98 },
  { name: 'Mathias', score: 80 },
  { name: 'Elias', score: 50 },
  { name: 'Martha', score: 85 },
  { name: 'John', score: 100 },
]

const scoresGreaterEighty = scores.filter((score) =>
score.score > 80)
console.log(scoresGreaterEighty)
```

```
[{name: 'Asabeneh', score: 95}, { name: 'Lidiya', score: 98
},{name: 'Martha', score: 85},{name: 'John', score: 100}]
```

reduce

reduce: Reduce takes a callback function. The call back function takes accumulator, current, and optional initial value as a parameter and returns a single value. It is a good practice to define an initial value for the accumulator value. If we do not specify this parameter, by default accumulator will get array first value. If our array is an *empty array*, then Javascript will throw an error.

```
arr.reduce((acc, cur) => {
  // some operations goes here before returning a value
 return
}, initialValue)
```

```
const numbers = [1, 2, 3, 4, 5]
const sum = numbers.reduce((acc, cur) => acc + cur, 0)

console.log(sum)
```

```
15
```

every

every: Check if all the elements are similar in one aspect. It returns boolean

```
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const areAllStr = names.every((name) => typeof name === 
'string') // Are all strings?

console.log(areAllStr)
```

```
true
```

```
const bools = [true, true, true, true]
const areAllTrue = bools.every((b) => b === true) // Are all 
true?

console.log(areAllTrue) // true
```

```
true
```

find

find: Return the first element which satisfies the condition

```
const ages = [24, 22, 25, 32, 35, 18]
const age = ages.find((age) => age < 20)

console.log(age)
```

```
18
```

```
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const result = names.find((name) => name.length > 7)
console.log(result)
```

```
Asabeneh
```

```
const scores = [
  { name: 'Asabeneh', score: 95 },
  { name: 'Mathias', score: 80 },
  { name: 'Elias', score: 50 },
  { name: 'Martha', score: 85 },
  { name: 'John', score: 100 },
]

const score = scores.find((user) => user.score > 80)
console.log(score)
{ name: "Asabeneh", score: 95 }
```

findIndex

findIndex: Return the position of the first element which satisfies the condition

```
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const ages = [24, 22, 25, 32, 35, 18]

const result = names.findIndex((name) => name.length > 7)
console.log(result) // 0

const age = ages.findIndex((age) => age < 20)
console.log(age) // 5
```

some

some: Check if some of the elements are similar in one aspect. It returns boolean

```
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
```

```
const bools = [true, true, true, true]

const areSomeTrue = bools.some((b) =>  b === true)

console.log(areSomeTrue) //true
```

```
const areAllStr = names.some((name) => typeof name ===
'number') // Are all strings ?
console.log(areAllStr) // false
```

sort

sort: The sort methods arranges the array elements either ascending or descending order. By default, the ***sort()*** method sorts values as strings.This works well for string array items but not for numbers. If number values are sorted as strings and it give us wrong result. Sort method modify the original array. It is recommended to copy the original data before you start using *sort* method.

Sorting string values

```
const products = ['Milk', 'Coffee', 'Sugar', 'Honey', 'Apple',
'Carrot']
console.log(products.sort()) // ['Apple', 'Carrot', 'Coffee',
'Honey', 'Milk', 'Sugar']
//Now the original products array  is also sorted
```

Sorting Numeric values

As you can see in the example below, 100 came first after sorted in ascending order. Sort converts items to string , since '100' and other numbers compared, 1 which the beginning of the string '100' became the smallest. To avoid this, we use a compare call back function inside the sort method, which return a negative, zero or positive.

```
const numbers = [9.81, 3.14, 100, 37]
// Using sort method to sort number items provide a wrong
result. see below
console.log(numbers.sort()) //[100, 3.14, 37, 9.81]
numbers.sort(function (a, b) {
  return a - b
})

console.log(numbers) // [3.14, 9.81, 37, 100]
```

```
numbers.sort(function (a, b) {
  return b - a
})
console.log(numbers) //[100, 37, 9.81, 3.14]
```

Sorting Object Arrays

Whenever we sort objects in an array, we use the object key to compare. Let us see the example below.

```
objArr.sort(function (a, b) {
  if (a.key < b.key) return -1
  if (a.key > b.key) return 1
  return 0
})

// or

objArr.sort(function (a, b) {
  if (a['key'] < b['key']) return -1
  if (a['key'] > b['key']) return 1
  return 0
})

const users = [
  { name: 'Asabeneh', age: 150 },
  { name: 'Brook', age: 50 },
  { name: 'Eyob', age: 100 },
  { name: 'Elias', age: 22 },
]
users.sort((a, b) => {
  if (a.age < b.age) return -1
  if (a.age > b.age) return 1
  return 0
})
console.log(users) // sorted ascending
// [{…}, {…}, {…}, {…}]
```

 You are doing great.Never give up because great things take time. You have just completed day 9 challenges and you are 9 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

💻 Exercises

Exercises: Level 1

```
const countries = ['Finland', 'Sweden', 'Denmark', 'Norway',
'IceLand']
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
const products = [
  { product: 'banana', price: 3 },
  { product: 'mango', price: 6 },
  { product: 'potato', price: ' ' },
  { product: 'avocado', price: 8 },
  { product: 'coffee', price: 10 },
  { product: 'tea', price: '' },
]
```

1. Explain the difference between ***forEach, map, filter, and reduce***.
2. Define a callback function before you use it in forEach, map, filter or reduce.
3. Use ***forEach*** to console.log each country in the countries array.
4. Use ***forEach*** to console.log each name in the names array.
5. Use ***forEach*** to console.log each number in the numbers array.
6. Use ***map*** to create a new array by changing each country to uppercase in the countries array.
7. Use ***map*** to create an array of countries length from countries array.
8. Use ***map*** to create a new array by changing each number to square in the numbers array
9. Use ***map*** to change to each name to uppercase in the names array
10. Use ***map*** to map the products array to its corresponding prices.
11. Use ***filter*** to filter out countries containing ***land***.
12. Use ***filter*** to filter out countries having six character.
13. Use ***filter*** to filter out countries containing six letters and more in the country array.
14. Use ***filter*** to filter out country start with 'E';
15. Use ***filter*** to filter out only prices with values.

16. Declare a function called getStringLists which takes an array as a parameter and then returns an array only with string items.
17. Use ***reduce*** to sum all the numbers in the numbers array.
18. Use ***reduce*** to concatenate all the countries and to produce this sentence: ***Estonia, Finland, Sweden, Denmark, Norway, and IceLand are north European countries***
19. Explain the difference between ***some*** and ***every***
20. Use ***some*** to check if some names' length greater than seven in names array
21. Use ***every*** to check if all the countries contain the word land
22. Explain the difference between ***find*** and ***findIndex***.
23. Use ***find*** to find the first country containing only six letters in the countries array
24. Use ***findIndex*** to find the position of the first country containing only six letters in the countries array
25. Use ***findIndex*** to find the position of ***Norway*** if it doesn't exist in the array you will get -1.
26. Use ***findIndex*** to find the position of ***Russia*** if it doesn't exist in the array you will get -1.

Exercises: Level 2

1. Find the total price of products by chaining two or more array iterators(eg. arr.map(callback).filter(callback).reduce(callback))
2. Find the sum of price of products using only reduce reduce(callback))
3. Declare a function called ***categorizeCountries*** which returns an array of countries which have some common pattern(you find the countries array in this repository as countries.js(eg 'land', 'ia', 'island','stan')).
4. Create a function which return an array of objects, which is the letter and the number of times the letter use to start with a name of a country.

5. Declare a ***getFirstTenCountries*** function and return an array of ten countries. Use different functional programming to work on the countries.js array
6. Declare a ***getLastTenCountries*** function which which returns the last ten countries in the countries array.
7. Find out which *letter* is used many *times* as initial for a country name from the countries array (eg. Finland, Fiji, France etc)

Exercises: Level 3

1. Use the countries information, in the data folder. Sort countries by name, by capital, by population
2. *** Find the 10 most spoken languages:

```
// Your output should look like this
console.log(mostSpokenLanguages(countries, 10))
[
{country: 'English',count:91},
{country: 'French',count:45},
{country: 'Arabic',count:25},
{country: 'Spanish',count:24},
{country:'Russian',count:9},
{country:'Portuguese', count:9},
{country:'Dutch',count:8},
{country:'German',count:7},
{country:'Chinese',count:5},
{country:'Swahili',count:4}
]

// Your output should look like this
console.log(mostSpokenLanguages(countries, 3))
[
{country: 'English',count: 91},
{country: 'French',count: 45},
{country: 'Arabic',count: 25},
]```
```

3. *** Use countries_data.js file create a function which create the ten most populated countries

```
console.log(mostPopulatedCountries(countries, 10))

[
{country: 'China', population: 1377422166},
```

```
{country: 'India', population: 1295210000},
{country: 'United States of America', population:
323947000},
{country: 'Indonesia', population: 258705000},
{country: 'Brazil', population: 206135893},
{country: 'Pakistan', population: 194125062},
{country: 'Nigeria', population: 186988000},
{country: 'Bangladesh', population: 161006790},
{country: 'Russian Federation', population: 146599183},
{country: 'Japan', population: 126960000}
]

console.log(mostPopulatedCountries(countries, 3))
[
{country: 'China', population: 1377422166},
{country: 'India', population: 1295210000},
{country: 'United States of America', population:
323947000}
]
```
```

4. *** Try to develop a program which calculate measure of central tendency of a sample(mean, median, mode) and measure of variability(range, variance, standard deviation). In addition to those measures find the min, max, count, percentile, and frequency distribution of the sample. You can create an object called statistics and create all the functions which do statistical calculations as method for the statistics object. Check the output below.

```
const ages = [31, 26, 34, 37, 27, 26, 32, 32, 26, 27, 27,
24, 32, 33, 27, 25, 26, 38, 37, 31, 34, 24, 33, 29, 26]

console.log('Count:', statistics.count()) // 25
console.log('Sum: ', statistics.sum()) // 744
console.log('Min: ', statistics.min()) // 24
console.log('Max: ', statistics.max()) // 38
console.log('Range: ', statistics.range() // 14
console.log('Mean: ', statistics.mean()) // 30
console.log('Median: ',statistics.median()) // 29
console.log('Mode: ', statistics.mode()) // {'mode': 26,
'count': 5}
console.log('Variance: ',statistics.var()) // 17.5
console.log('Standard Deviation: ', statistics.std()) //
4.2
console.log('Variance: ',statistics.var()) // 17.5
console.log('Frequency Distribution:
',statistics.freqDist()) # [(20.0, 26), (16.0, 27),
(12.0, 32), (8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31),
(8.0, 24), (4.0, 38), (4.0, 29), (4.0, 25)]
console.log(statistics.describe())
```
```

```
Count: 25
Sum:  744
Min:  24
Max:  38
Range:  14
Mean:  30
Median:  29
Mode:  (26, 5)
Variance:  17.5
Standard Deviation:  4.2
Frequency Distribution: [(20.0, 26), (16.0, 27), (12.0,
32), (8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31), (8.0,
24), (4.0, 38), (4.0, 29), (4.0, 25)]
```

 CONGRATULATIONS ! DAY-9 COMPLETED 

DAY-10 SETS AND MAPS

Set

Set is a collection of elements. Set can only contains unique elements. Let us see how to create a set in the section below.

Creating an empty set

```
const companies = new Set()
console.log(companies)
```

```
Set(0) {}
```

Creating set from array

```
const languages = [
  'English',
  'Finnish',
  'English',
  'French',
  'Spanish',
  'English',
  'French',
]

const setOfLanguages = new Set(languages)
console.log(setOfLanguages)
```

```
Set(4) {"English", "Finnish", "French", "Spanish"}
```

Set is an iterable object and we can iterate through each elements.

```
const languages = [
  'English',
  'Finnish',
  'English',
  'French',
  'Spanish',
  'English',
  'French',
]

const setOfLanguages = new Set(languages)
```

```
for (const language of setOfLanguages) {
  console.log(language)
}
```

```
  English
  Finnish
  French
  Spanish
```

Adding an element to a set

```
const companies = new Set() // creating an empty set
console.log(companies.size) // 0

companies.add('Google') // add element to the set
companies.add('Facebook')
companies.add('Amazon')
companies.add('Oracle')
companies.add('Microsoft')
console.log(companies.size) // 5 elements in the set
console.log(companies)
```

```
Set(5) {"Google", "Facebook", "Amazon", "Oracle", "Microsoft"}
```

We can also use loop to add element to a set.

```
const companies = ['Google', 'Facebook', 'Amazon', 'Oracle',
'Microsoft']
setOfCompanies = new Set()
for (const company of companies) {
  setOfCompanies.add(company)
}
Set(5) {"Google", "Facebook", "Amazon", "Oracle", "Microsoft"}
```

Deleting an element a set

We can delete an element from a set using a delete method.

```
console.log(companies.delete('Google'))
console.log(companies.size) // 4 elements left in the set
```

Checking an element in the set

The has method can help to know if a certain element exists in a set.

```
console.log(companies.has('Apple')) // false
console.log(companies.has('Facebook')) // true
```

Clearing the set

It removes all the elements from a set.

```
companies.clear()
console.log(companies)
```

```
Set(0) {}
```

See the example below to learn how to use set.

```
const languages = [
  'English',
  'Finnish',
  'English',
  'French',
  'Spanish',
  'English',
  'French',
]
const langSet = new Set(languages)
console.log(langSet) // Set(4) {"English", "Finnish",
"French", "Spanish"}
console.log(langSet.size) // 4

const counts = []
const count = {}

for (const l of langSet) {
  const filteredLang = languages.filter((lng) => lng === l)
  console.log(filteredLang) // ["English", "English",
"English"]
  counts.push({ lang: l, count: filteredLang.length })
}
console.log(counts)
```

```
[
  { lang: 'English', count: 3 },
  { lang: 'Finnish', count: 1 },
  { lang: 'French', count: 2 },
```

```
  { lang: 'Spanish', count: 1 },
]
```

Other use case of set. For instance to count unique item in an array.

```
const numbers = [5, 3, 2, 5, 5, 9, 4, 5]
const setOfNumbers = new Set(numbers)

console.log(setOfNumbers)
```

```
Set(5) {5, 3, 2, 9, 4}
```

Union of sets

To find a union to two sets can be achieved using spread operator. Lets find the union of set A and set B (A U B)

```
let a = [1, 2, 3, 4, 5]
let b = [3, 4, 5, 6]
let c = [...a, ...b]

let A = new Set(a)
let B = new Set(b)
let C = new Set(c)

console.log(C)
```

```
Set(6) {1, 2, 3, 4, 5,6}
```

Intersection of sets

To find an intersection of two sets can be achieved using filter. Lets find the intersection of set A and set B (A ∩ B)

```
let a = [1, 2, 3, 4, 5]
let b = [3, 4, 5, 6]

let A = new Set(a)
let B = new Set(b)

let c = a.filter((num) => B.has(num))
let C = new Set(c)

console.log(C)
```

```
Set(3) {3, 4, 5}
```

Difference of sets

To find an the difference between two sets can be achieved using filter. Lets find the different of set A and set B (A \ B)

```
let a = [1, 2, 3, 4, 5]
let b = [3, 4, 5, 6]

let A = new Set(a)
let B = new Set(b)

let c = a.filter((num) => !B.has(num))
let C = new Set(c)

console.log(C)
```

```
Set(2) {1, 2}
```

Map

Creating an empty Map

```
const map = new Map()
console.log(map)
```

```
Map(0) {}
```

Creating an Map from array

```
countries = [
  ['Finland', 'Helsinki'],
  ['Sweden', 'Stockholm'],
  ['Norway', 'Oslo'],
]
const map = new Map(countries)
console.log(map)
console.log(map.size)
```

```
Map(3) {"Finland" => "Helsinki", "Sweden" => "Stockholm",
"Norway" => "Oslo"}
3
```

Adding values to the Map

```
const countriesMap = new Map()
console.log(countriesMap.size) // 0
countriesMap.set('Finland', 'Helsinki')
countriesMap.set('Sweden', 'Stockholm')
countriesMap.set('Norway', 'Oslo')
console.log(countriesMap)
console.log(countriesMap.size)
```

```
Map(3) {"Finland" => "Helsinki", "Sweden" => "Stockholm",
"Norway" => "Oslo"}
3
```

Getting a value from Map

```
console.log(countriesMap.get('Finland'))
Helsinki
```

Checking key in Map

Check if a key exists in a map using *has* method. It returns *true* or *false*.

```
console.log(countriesMap.has('Finland'))
true
```

Getting all values from map using loop

```
for (const country of countriesMap) {
  console.log(country)
}
```

```
(2) ["Finland", "Helsinki"]
(2) ["Sweden", "Stockholm"]
(2) ["Norway", "Oslo"]
```

```
for (const [country, city] of countriesMap){
 console.log(country, city)
}
```

```
Finland Helsinki
Sweden Stockholm
Norway Oslo
```

🥲 You established a big milestone, you are unstoppable. Keep going! You have just completed day 10 challenges and you are 10 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises:Level 1

```
const a = [4, 5, 8, 9]
const b = [3, 4, 5, 7]
const countries = ['Finland', 'Sweden', 'Norway']
```

1. create an empty set
2. Create a set containing 0 to 10 using loop
3. Remove an element from a set
4. Clear a set
5. Create a set of 5 string elements from array
6. Create a map of countries and number of characters of a country

Exercises:Level 2

1. Find a union b
2. Find a intersection b
3. Find a with b

Exercises:Level 3

1. How many languages are there in the countries object file.
2. *** Use the countries data to find the 10 most spoken languages:

```
// Your output should look like this
console.log(mostSpokenLanguages(countries, 10))
[
  { English: 91 },
  { French: 45 },
  { Arabic: 25 },
```

```
    { Spanish: 24 },
    { Russian: 9 },
    { Portuguese: 9 },
    { Dutch: 8 },
    { German: 7 },
    { Chinese: 5 },
    { Swahili: 4 },
    { Serbian: 4 }
 ]

// Your output should look like this
console.log(mostSpokenLanguages(countries, 3))
[
{English:91},
{French:45},
{Arabic:25}
]
```

 CONGRATULATIONS ! DAY-10 COMPLETED 

DAY-11 DESTRUCTURING AND SPREADING

Destructuring is a way to unpack arrays, and objects and assigning to a distinct variable.

Destructing Arrays

```
  const numbers = [1, 2, 3]
  let [numOne, numTwo, numThree] = numbers

  console.log(numOne, numTwo, numThree)
```

```
 1 2 3
```

```
  const names = ['Asabeneh', 'Brook', 'David', 'John']
  let [firstPerson, secondPerson, thirdPerson, fourthPerson] =
names

  console.log(firstPerson, secondPerson,thirdPerson,
fourthPerson)
```

```
Asabeneh Brook David John
```

```
  const scientificConstants = [2.72, 3.14, 9.81, 37, 100]
  let [e, pi, gravity, bodyTemp, boilingTemp] =
scientificConstants

  console.log(e,pi,gravity, bodyTemp, boilingTemp)
```

```
2.72 3.14 9.81 37 100

const fullStack = [
  ['HTML', 'CSS', 'JS', 'React'],
  ['Node', 'Express', 'MongoDB']
]
const [frontEnd, backEnd] = fullStack

console.log(frontEnd)
console.log(backEnd)
```

```
["HTML", "CSS", "JS", "React"]
["Node", "Express", "MongoDB"]
```

If we like to skip on of the values in the array we use additional comma. The comma helps to omit the value at that specific index

```
  const numbers = [1, 2, 3]
  let [numOne, , numThree] = numbers //2 is omitted

  console.log(numOne, numThree)
```

```
1 3
```

```
  const names = ['Asabeneh', 'Brook', 'David', 'John']
  let [, secondPerson, , fourthPerson] = names // first and
third person is omitted

  console.log(secondPerson, fourthPerson)
```

```
Brook John
```

We can use default value in case the value of array for that index is undefined:

```
const names = [undefined, 'Brook', 'David']
let [
  firstPerson = 'Asabeneh',
  secondPerson,
  thirdPerson,
  fourthPerson = 'John'
] = names

console.log(firstPerson, secondPerson, thirdPerson,
fourthPerson)
```

```
Asabeneh Brook David John
```

We can not assign variable to all the elements in the array. We can destructure few of the first and we can get the remaining as array using spread operator(...).

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
let [num1, num2, num3, ...rest] = nums

console.log(num1, num2, num3)
console.log(rest)
```

```
1 2 3
[4, 5, 6, 7, 8, 9, 10]
```

Destructuring during iteration

```
const countries = [['Finland', 'Helsinki'], ['Sweden',
'Stockholm'], ['Norway', 'Oslo']]

for (const [country, city] of countries) {
console.log(country, city)
}
Finland Helsinki
Sweden Stockholm
Norway Oslo
```

```
const fullStack = [
  ['HTML', 'CSS', 'JS', 'React'],
  ['Node', 'Express', 'MongoDB']
]

for(const [first, second, third] of fullStack) {
console.log(first, second, third)
}
```

```
HTML CSS JS
Node Express MongoDB
```

Destructuring Object

When we destructure the name of the variable we use to destructure should be exactly the same as the key or property of the object. See the example below.

```
const rectangle = {
  width: 20,
  height: 10,
  area: 200
}
let { width, height, area, perimeter } = rectangle

console.log(width, height, area, perimeter)
```

```
20 10 200 undefined
```

Renaming during structuring

```
const rectangle = {
  width: 20,
  height: 10,
  area: 200
}
let { width: w, height: h, area: a, perimeter: p } = rectangle

console.log(w, h, a, p)
```

```
20 10 200 undefined
```

If the key is not found in the object the variable will be assigned to undefined. Sometimes the key might not be in the object, in that case we can give a default value during declaration. See the example.

```
const rectangle = {
  width: 20,
  height: 10,
  area: 200
}
let { width, height, area, perimeter = 60 } = rectangle

console.log(width, height, area, perimeter) //20 10 200 60
//Let us modify the object:width to 30 and perimeter to 80
const rectangle = {
  width: 30,
  height: 10,
  area: 200,
  perimeter: 80
}
let { width, height, area, perimeter = 60 } = rectangle
console.log(width, height, area, perimeter) //30 10 200 80
```

Destructuring keys as a function parameters. Let us create a function which takes a rectangle object and it returns a perimeter of a rectangle.

Object parameter without destructuring

```
// Without destructuring
const rect = {
  width: 20,
  height: 10
```

```
}
const calculatePerimeter = rectangle => {
  return 2 * (rectangle.width + rectangle.height)
}

console.log(calculatePerimeter(rect)) // 60
//with destructuring
```

```
//Another Example
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  job: 'Instructor and Developer',
  skills: [
    'HTML',
    'CSS',
    'JavaScript',
    'React',
    'Redux',
    'Node',
    'MongoDB',
    'Python',
    'D3.js'
  ],
  languages: ['Amharic', 'English', 'Suomi(Finnish)']
}
// Let us create a function which give information about the
person object without destructuring

const getPersonInfo = obj => {
  const skills = obj.skills
  const formattedSkills = skills.slice(0, -1).join(', ')
  const languages = obj.languages
  const formattedLanguages = languages.slice(0, -1).join(', ')

  personInfo = `${obj.firstName} ${obj.lastName} lives in
${obj.country}. He is  ${
    obj.age
  } years old. He is an ${obj.job}. He teaches
${formattedSkills} and ${
    skills[skills.length - 1]
  }. He speaks ${formattedLanguages} and a little bit of
${languages[2]}.`

  return personInfo
}
```

```
console.log(getPersonInfo(person))
```

Object parameter with destructuring

```
const calculatePerimeter = ({ width, height }) => {
  return 2 * (width + height)
}

console.log(calculatePerimeter(rect)) // 60
```

```
// Let us create a function which give information about the
person object with destructuring
const getPersonInfo = ({
  firstName,
  lastName,
  age,
  country,
  job,
  skills,
  languages
}) => {
  const formattedSkills = skills.slice(0, -1).join(', ')
  const formattedLanguages = languages.slice(0, -1).join(', ')

  personInfo = `${firstName} ${lastName} lives in ${country}.
He is ${age} years old. He is an ${job}. He teaches
${formattedSkills} and ${
    skills[skills.length - 1]
  }. He speaks ${formattedLanguages} and a little bit of
${languages[2]}.`

  return personInfo
}
console.log(getPersonInfo(person))
/*
Asabeneh Yetayeh lives in Finland. He is  250 years old. He is
an Instructor and Developer. He teaches HTML, CSS, JavaScript,
React, Redux, Node, MongoDB, Python and D3.js. He speaks
Amharic, English and a little bit of Suomi(Finnish)
*/
```

Destructuring object during iteration

```
const todoList = [
{
  task:'Prepare JS Test',
  time:'4/1/2020 8:30',
  completed:true
},
{
  task:'Give JS Test',
  time:'4/1/2020 10:00',
  completed:false
},
{
  task:'Assess Test Result',
  time:'4/1/2020 1:00',
  completed:false
}
]

for (const {task, time, completed} of todoList){
  console.log(task, time, completed)
}
```

```
Prepare JS Test 4/1/2020 8:30 true
Give JS Test 4/1/2020 10:00 false
Assess Test Result 4/1/2020 1:00 false
```

Spread or Rest Operator

When we destructure an array we use the spread operator(...) to get the rest elements as array. In addition to that we use spread operator to spread array elements to another array.

Spread operator to get the rest of array elements

```
const nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
let [num1, num2, num3, ...rest] = nums

console.log(num1, num2, num3)
console.log(rest)
```

```
1 2 3
[4, 5, 6, 7, 8, 9, 10]
```

```
const countries = [
  'Germany',
  'France',
  'Belgium',
  'Finland',
  'Sweden',
  'Norway',
  'Denmark',
  'Iceland'
]

let [gem, fra, , ...nordicCountries] = countries

console.log(gem)
console.log(fra)
console.log(nordicCountries)
```

```
Germany
France
["Finland", "Sweden", "Norway", "Denmark", "Iceland"]
```

Spread operator to copy array

```
const evens = [0, 2, 4, 6, 8, 10]
const evenNumbers = [...evens]

const odds = [1, 3, 5, 7, 9]
const oddNumbers = [...odds]

const wholeNumbers = [...evens, ...odds]

console.log(evenNumbers)
console.log(oddNumbers)
console.log(wholeNumbers)
[0, 2, 4, 6, 8, 10]
[1, 3, 5, 7, 9]
[0, 2, 4, 6, 8, 10, 1, 3, 5, 7, 9]
```

```
const frontEnd = ['HTML', 'CSS', 'JS', 'React']
const backEnd = ['Node', 'Express', 'MongoDB']
const fullStack = [...frontEnd, ...backEnd]

console.log(fullStack)
```

```
["HTML", "CSS", "JS", "React", "Node", "Express", "MongoDB"]
```

Spread operator to copy object

We can copy an object using a spread operator

```
const user = {
  name:'Asabeneh',
  title:'Programmer',
  country:'Finland',
  city:'Helsinki'
}

const copiedUser = {...user}
console.log(copiedUser)
```

```
{name: "Asabeneh", title: "Programmer", country: "Finland",
city: "Helsinki"}
```

Modifying or changing the object while copying

```
const user = {
  name:'Asabeneh',
  title:'Programmer',
  country:'Finland',
  city:'Helsinki'
}

const copiedUser = {...user, title:'instructor'}
console.log(copiedUser)
```

```
{name: "Asabeneh", title: "instructor", country: "Finland",
city: "Helsinki"}
```

Spread operator with arrow function

Whenever we like to write an arrow function which takes unlimited number of arguments we use a spread operator. If we use a spread operator as a parameter, the argument passed when we invoke a function will change to an array.

```
const sumAllNums = (...args) => {
  console.log(args)
}

sumAllNums(1, 2, 3, 4, 5)
```

```
[1, 2, 3, 4, 5]
```

```
const sumAllNums = (...args) => {
  let sum = 0
  for (const num of args){
    sum += num
  }
  return sum

}

console.log(sumAllNums(1, 2, 3, 4, 5))

15
```

 You achieved quite a lot so far. Now, your level of JavaScript is upper intermediate. Keep going! You have just completed day 11 challenges and you are 11 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises: Level 1

```
const constants = [2.72, 3.14, 9.81, 37, 100]
const countries = ['Finland', 'Estonia', 'Sweden', 'Denmark',
'Norway']
const rectangle = {
  width: 20,
  height: 10,
  area: 200,
  perimeter: 60
}
const users = [
{
  name:'Brook',
  scores:75,
  skills:['HTM', 'CSS', 'JS'],
  age:16
},
{
  name:'Alex',
  scores:80,
  skills:['HTM', 'CSS', 'JS'],
  age:18
},
{
  name:'David',
  scores:75,
  skills:['HTM', 'CSS'],
  age:22
},
{
  name:'John',
  scores:85,
  skills:['HTML'],
  age:25
},
{
  name:'Sara',
  scores:95,
  skills:['HTM', 'CSS', 'JS'],
  age: 26
},
{
  name:'Martha',
  scores:80,
  skills:['HTM', 'CSS', 'JS'],
  age:18
```

```
},
{
  name:'Thomas',
  scores:90,
  skills:['HTM', 'CSS', 'JS'],
  age:20
}
]
```

1. Destructure and assign the elements of constants array to e, pi, gravity, humanBodyTemp, waterBoilingTemp.
2. Destructure and assign the elements of countries array to fin, est, sw, den, nor
3. Destructure the rectangle object by its properties or keys.

Exercises: Level 2

1. Iterate through the users array and get all the keys of the object using destructuring
2. Find the persons who have less than two skills

Exercises: Level 3

1. Destructure the countries object print name, capital, population and languages of all countries
2. A junior developer structure student name, skills and score in array of arrays which may not easy to read. Destructure the following array name to name, skills array to skills, scores array to scores, JavaScript score to jsScore and React score to reactScore variable in one line.

```
  const student = ['David', ['HTM', 'CSS', 'JS', 'React'],
[98, 85, 90, 95]]
  console.log(name, skills, jsScore, reactScore)
```

```
David (4) ["HTM", "CSS", "JS", "React"] 90 95
```

3. Write a function called *convertArrayToObject* which can convert the array to a structure object.

```
    const students = [
        ['David', ['HTM', 'CSS', 'JS', 'React'], [98, 85, 90,
95]],
        ['John', ['HTM', 'CSS', 'JS', 'React'], [85, 80, 85,
80]]
      ]

    console.log(convertArrayToObject(students))
    [
      {
        name: 'David',
        skills: ['HTM','CSS','JS','React'],
        scores: [98,85,90,95]
      },
      {
        name: 'John',
        skills: ['HTM','CSS','JS','React'],
        scores: [85, 80,85,80]
      }
    ]
```

4. Copy the student object to newStudent without mutating the original object. In the new object add the following ?

- Add Bootstrap with level 8 to the front end skill sets
- Add Express with level 9 to the back end skill sets
- Add SQL with level 8 to the data base skill sets
- Add SQL without level to the data science skill sets

```
    const student = {
      name: 'David',
      age: 25,
      skills: {
        frontEnd: [
          { skill: 'HTML', level: 10 },
          { skill: 'CSS', level: 8 },
          { skill: 'JS', level: 8 },
          { skill: 'React', level: 9 }
        ],
        backEnd: [
          { skill: 'Node',level: 7 },
          { skill: 'GraphQL', level: 8 },
        ],
```

```
        dataBase:[
          { skill: 'MongoDB', level: 7.5 },
        ],
        dataScience:['Python', 'R', 'D3.js']
      }
    }
```

The copied object output should look like this:

```
    {
    name: 'David',
    age: 25,
    skills: {
      frontEnd: [
        {skill: 'HTML',level: 10},
        {skill: 'CSS',level: 8},
        {skill: 'JS',level: 8},
        {skill: 'React',level: 9},
        {skill: 'BootStrap',level: 8}
      ],
      backEnd: [
        {skill: 'Node',level: 7},
        {skill: 'GraphQL',level: 8},
        {skill: 'Express',level: 9}
      ],
      dataBase: [
        { skill: 'MongoDB',level: 7.5},
        { skill: 'SQL',level: 8}
      ],
      dataScience: ['Python','R','D3.js','SQL']
    }
  }
```

🎉 CONGRATULATIONS ! DAY 11 COMPLETED 🎉

DAY-12 REGULAR EXPRESSION

Regular Expressions

A regular expression or RegExp is a small programming language that helps to find pattern in data. A RegExp can be used to check if some pattern exists in a different data types. To use RegExp in JavaScript either we use RegExp constructor or we can declare a RegExp pattern using two forward slashes followed by a flag. We can create a pattern in two ways.

To declare a string we use a single quote, double quote a backtick to declare a regular expression we use two forward slashes and an optional flag. The flag could be g, i, m, s, u or y.

RegExp parameters

A regular expression takes two parameters. One required search pattern and an optional flag.

Pattern

A pattern could be a text or any form of pattern which some sort of similarity. For instance the word spam in an email could be a pattern we are interested to look for in an email or a phone number format number might be our interest to look for.

Flags

Flags are optional parameters in a regular expression which determine the type of searching. Let us see some of the flags:

- g: a global flag which means looking for a pattern in whole text
- i: case insensitive flag(it searches for both lowercase and uppercase)
- m: multiline

Creating a pattern with RegExp Constructor

Declaring regular expression without global flag and case insensitive flag.

```
// without flag
let pattern = 'love'
let regEx = new RegExp(pattern)
```

Declaring regular expression with global flag and case insensitive flag.

```
let pattern = 'love'
let flag = 'gi'
let regEx = new RegExp(pattern, flag)
```

Declaring a regex pattern using RegExp object. Writing the pattern and the flag inside the RegExp constructor

```
let regEx = new RegExp('love','gi')
```

Creating a pattern without RegExp Constructor

Declaring regular expression with global flag and case insensitive flag.

```
let regEx= /love/gi
```

The above regular expression is the same as the one which we created with RegExp constructor

```
let regEx= new RegExp('love','gi')
```

RegExpp Object Methods

Let us see some of RegExp methods

Testing for a match

test():Tests for a match in a string. It returns true or false.

```
const str = 'I love JavaScript'
const pattern = /love/
const result = pattern.test(str)
console.log(result)

true
```

Array containing all of the match

match():Returns an array containing all of the matches, including capturing groups, or null if no match is found. If we do not use a global flag, match() returns an array containing the pattern, index, input and group.

```
const str = 'I love JavaScript'
const pattern = /love/
const result = str.match(pattern)
console.log(result)
```

```
["love", index: 2, input: "I love JavaScript", groups:
undefined]
```

```
const str = 'I love JavaScript'
const pattern = /love/g
const result = str.match(pattern)
console.log(result)

["love"]
```

search(): Tests for a match in a string. It returns the index of the match, or -1 if the search fails.

```
const str = 'I love JavaScript'
const pattern = /love/g
const result = str.search(pattern)
console.log(result)

2
```

Replacing a substring

replace(): Executes a search for a match in a string, and replaces the matched substring with a replacement substring.

```
const txt = 'Python is the most beautiful language that a
human begin has ever created.\
I recommend python for a first programming language'

matchReplaced = txt.replace(/Python|python/, 'JavaScript')
console.log(matchReplaced)
JavaScript is the most beautiful language that a human begin
has ever created.I recommend python for a first programming
language
```

```
const txt = 'Python is the most beautiful language that a
human begin has ever created.\
I recommend python for a first programming language'

matchReplaced = txt.replace(/Python|python/g, 'JavaScript')
console.log(matchReplaced)
```

```
JavaScript is the most beautiful language that a human begin
has ever created.I recommend JavaScript for a first
programming language
```

```
const txt = 'Python is the most beautiful language that a
human begin has ever created.\
I recommend python for a first programming language'

matchReplaced = txt.replace(/Python/gi, 'JavaScript')
console.log(matchReplaced)
```

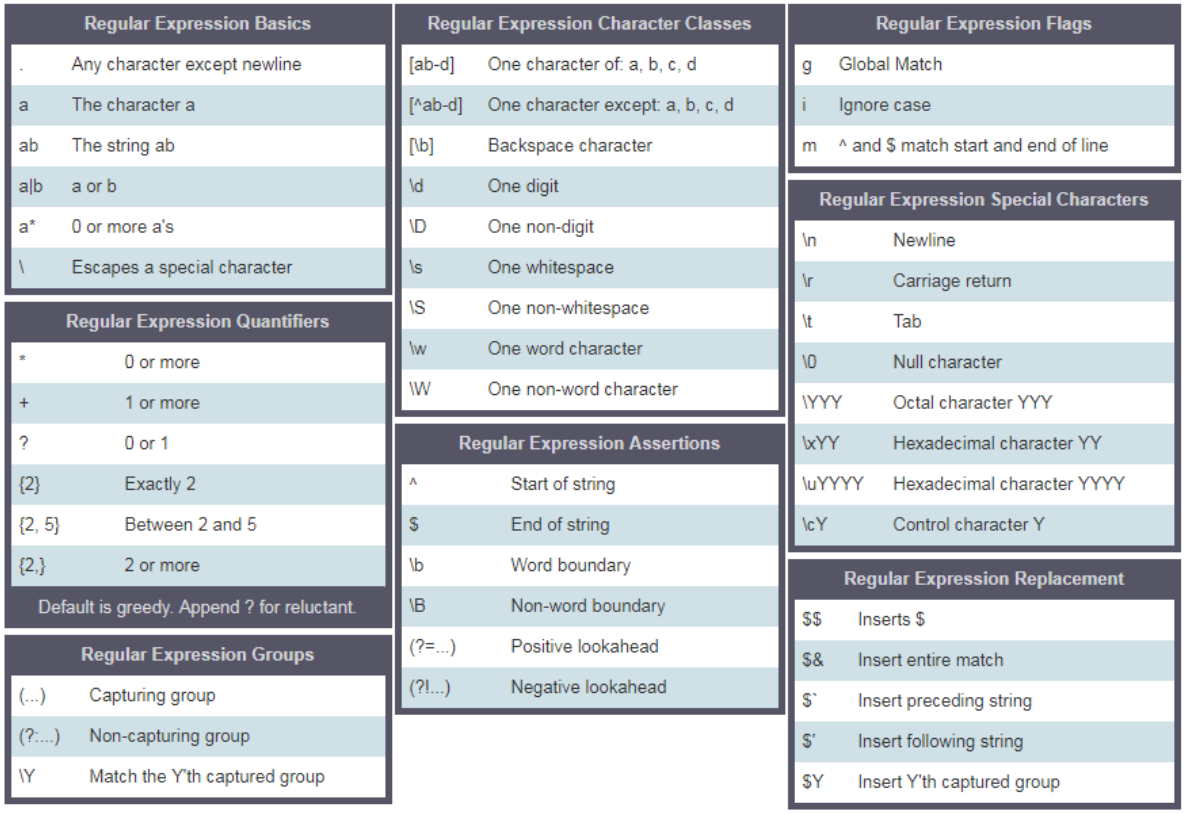
```
JavaScript is the most beautiful language that a human begin
has ever created.I recommend JavaScript for a first
programming language
const txt = '%I a%m te%%a%%che%r% a%n%d %% I l%o%ve
te%ach%ing.\
T%he%re i%s n%o%th%ing as m%ore r%ewarding a%s e%duc%at%i%ng
a%n%d e%m%p%ow%er%ing \
p%e%o%ple.\
I fo%und te%a%ching m%ore i%n%t%er%%es%ting t%h%an any other
%jobs.\
D%o%es thi%s m%ot%iv%a%te %y%o%u to b%e a t%e%a%cher.'
```

```
matches = txt.replace(/%/g, '')
console.log(matches)
```

```
I am teacher and  I love teaching.There is nothing as more
rewarding as educating and empowering people.I found teaching
more interesting than any other jobs.Does this motivate you to
be a teacher.
```

- []: A set of characters
    - [a-c] means, a or b or c
    - [a-z] means, any letter a to z
    - [A-Z] means, any character A to Z
    - [0-3] means, 0 or 1 or 2 or 3
    - [0-9] means any number 0 to 9
    - [A-Za-z0-9] any character which is a to z, A to Z, 0 to 9
- \: uses to escape special characters
    - \d mean: match where the string contains digits (numbers from 0-9)
    - \D mean: match where the string does not contain digits
- . : any character except new line character(\n)
- ^: starts with
    - r'^substring' eg r'^love', a sentence which starts with a word love
    - r'[^abc] mean not a, not b, not c.
- $: ends with
    - r'substring$' eg r'love$', sentence ends with a word love
- *: zero or more times
    - r'[a]*' means a optional or it can occur many times.
- +: one or more times
    - r'[a]+' means at least once or more times
- ?: zero or one times

    - r'[a]?' means zero times or once
- \b: word bounder, matches with the beginning or ending of a word
- {3}: Exactly 3 characters
- {3,}: At least 3 characters
- {3,8}: 3 to 8 characters
- |: Either or
    - r'apple|banana' mean either of an apple or a banana
- (): Capture and group



Regular Expression Basics	
.	Any character except newline
a	The character a
ab	The string ab
a|b	a or b
a*	0 or more a's
\	Escapes a special character

Regular Expression Quantifiers	
*	0 or more
+	1 or more
?	0 or 1
{2}	Exactly 2
{2, 5}	Between 2 and 5
{2,}	2 or more

Default is greedy. Append ? for reluctant.

Regular Expression Groups	
(...)	Capturing group
(?:...)	Non-capturing group
\Y	Match the Y'th captured group

Regular Expression Character Classes	
[ab-d]	One character of: a, b, c, d
[^ab-d]	One character except: a, b, c, d
[\b]	Backspace character
\d	One digit
\D	One non-digit
\s	One whitespace
\S	One non-whitespace
\w	One word character
\W	One non-word character

Regular Expression Assertions	
^	Start of string
$	End of string
\b	Word boundary
\B	Non-word boundary
(?=...)	Positive lookahead
(?!...)	Negative lookahead

Regular Expression Flags	
g	Global Match
i	Ignore case
m	^ and $ match start and end of line

Regular Expression Special Characters	
\n	Newline
\r	Carriage return
\t	Tab
\0	Null character
\YYY	Octal character YYY
\xYY	Hexadecimal character YY
\uYYYY	Hexadecimal character YYYY
\cY	Control character Y

Regular Expression Replacement	
$$	Inserts $
$&	Insert entire match
$`	Insert preceding string
$'	Insert following string
$Y	Insert Y'th captured group

Let's use example to clarify the above meta characters

Square Bracket

Let's use square bracket to include lower and upper case

```
const pattern = '[Aa]pple' // this square bracket means either
A or a
```

```
const txt = 'Apple and banana are fruits. An old cliche says
an apple a day keeps the  doctor way has been replaced by a
banana a day keeps the doctor far far away. '
const matches = txt.match(pattern)

console.log(matches)
```

```
["Apple", index: 0, input: "Apple and banana are fruits. An
old cliche says an apple a day keeps the  doctor way has been
replaced by a banana a day keeps the doctor far far away.",
groups: undefined]
const pattern = /[Aa]pple/g // this square bracket means
either A or a
const txt = 'Apple and banana are fruits. An old cliche says
an apple a day a doctor way has been replaced by a banana a
day keeps the doctor far far away. '
const matches = txt.match(pattern)

console.log(matches)
```

```
["Apple", "apple"]
```

If we want to look for the banana, we write the pattern as follows:

```
const pattern = /[Aa]pple|[Bb]anana/g // this square bracket
mean either A or a
const txt = 'Apple and banana are fruits. An old cliche says
an apple a day a doctor way has been replaced by a banana a
day keeps the doctor far far away. Banana is easy to eat too.'
const matches = txt.match(pattern)

console.log(matches)
```

```
["Apple", "banana", "apple", "banana", "Banana"]
```

Using the square bracket and or operator , we manage to extract Apple, apple, Banana and banana.

Escape character(\) in RegExp

```
const pattern = /\d/g  // d is a special character which means
digits
const txt = 'This regular expression example was made in
January 12,  2020.'
const matches = txt. match(pattern)

console.log(matches)  // ["1", "2", "2", "0", "2", "0"], this
is not what we want
```

```
const pattern = /\d+/g  // d is a special character which
means digits
const txt = 'This regular expression example was made in
January 12,  2020.'
const matches = txt. match(pattern)

console.log(matches)  // ["12", "2020"], this is not what we
want
```

One or more times(+)

```
const pattern = /\d+/g  // d is a special character which
means digits
const txt = 'This regular expression example was made in
January 12,  2020.'
const matches = txt. match(pattern)
console.log(matches)  // ["12", "2020"], this is not what we
want
```

Period(.)

```
const pattern = /[a]./g  // this square bracket means a and .
means any character except new line
const txt = 'Apple and banana are fruits'
const matches = txt.match(pattern)
```

```
console.log(matches)  // ["an", "an", "an", "a ", "ar"]
const pattern = /[a].+/g  // . any character, + any character
one or more times
const txt = 'Apple and banana are fruits'
const matches = txt.match(pattern)

console.log(matches)  // ['and banana are fruits']
```

Zero or more times(*)

Zero or many times. The pattern may not occur or it can occur many times.

```
const pattern = /[a].*/g  //. any character, + any character
one or more times
const txt = 'Apple and banana are fruits'
const matches = txt.match(pattern)

console.log(matches)  // ['and banana are fruits']
```

Zero or one times(?)

Zero or one times. The pattern may not occur or it may occur once.

```
const txt = 'I am not sure if there is a convention how to
write the word e-mail.\
Some people write it email others may write it as Email or E-
mail.'
const pattern = /[Ee]-?mail/g  // ? means optional
matches = txt.match(pattern)

console.log(matches)  // ["e-mail", "email", "Email", "E-
mail"]
```

Quantifier in RegExp

We can specify the length of the substring we look for in a text, using a curly bracket. Let us see, how ot use RegExp quantifiers. Imagine, we are interested in substring that their length are 4 characters

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /\\b\w{4}\b/g  //  exactly four character
words
const matches = txt.match(pattern)
console.log(matches)  //['This', 'made', '2019']
```

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /\b[a-zA-Z]{4}\b/g  //  exactly four character
words without numbers
const matches = txt.match(pattern)
console.log(matches)  //['This', 'made']
```

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /\d{4}/g  // a number and exactly four digits
const matches = txt.match(pattern)
console.log(matches)  // ['2019']
```

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /\d{1,4}/g   // 1 to 4
const matches = txt.match(pattern)
console.log(matches)  // ['6', '2019']
```

Cart ^

- Starts with

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /^This/ // ^ means starts with
const matches = txt.match(pattern)
console.log(matches)  // ['This']
```

- Negation

```
const txt = 'This regular expression example was made in
December 6,  2019.'
const pattern = /[^A-Za-z,. ]+/g  // ^ in set character means
negation, not A to Z, not a to z, no space, no comma no period
const matches = txt.match(pattern)
console.log(matches)  // ["6", "2019"]
```

Exact match

It should have ^ starting and $ which is an end.

```
let pattern = /^[A-Z][a-z]{3,12}$/;
let name = 'Asabeneh';
let result = pattern.test(name)

console.log(result) // true
```

You are going far. Keep going! Now, you are super charged with the power of regular expression. You have the power to extract and clean any kind of text and you can make meaning out of unstructured data. You have just completed day 12 challenges and you are 12 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises: Level 1

1. Calculate the total annual income of the person from the following text. 'He earns 4000 euro from salary per month, 10000 euro annual bonus, 5500 euro online courses per month.'
2. The position of some particles on the horizontal x-axis -12, -4, -3 and -1 in the negative direction, 0 at origin, 4 and 8 in the positive direction. Extract these numbers and find the distance between the two furthest particles.

```
points = ['-1', '2', '-4', '-3', '-1', '0', '4', '8']
sortedPoints =  [-4, -3, -1, -1, 0, 2, 4, 8]
distance = 12
```

1. Write a pattern which identify if a string is a valid JavaScript variable

```
is_valid_variable('first_name') # True
is_valid_variable('first-name') # False
is_valid_variable('1first_name') # False
is_valid_variable('firstname') # True
```

Exercises: Level 2

1. Write a function called *tenMostFrequentWords* which get the ten most frequent word from a string?

```
paragraph = `I love teaching. If you do not love teaching
what else can you love. I love Python if you do not love
something which can give you all the capabilities to
develop an application what else can you love.`
    console.log(tenMostFrequentWords(paragraph))
```

```
[
   {word:'love', count:6},
   {word:'you', count:5},
   {word:'can', count:3},
   {word:'what', count:2},
   {word:'teaching', count:2},
   {word:'not', count:2},
   {word:'else', count:2},
   {word:'do', count:2},
   {word:'I', count:2},
   {word:'which', count:1},
   {word:'to', count:1},
   {word:'the', count:1},
```

```
    {word:'something', count:1},
    {word:'if', count:1},
    {word:'give', count:1},
    {word:'develop',count:1},
    {word:'capabilities',count:1},
    {word:'application', count:1},
    {word:'an',count:1},
    {word:'all',count:1},
    {word:'Python',count:1},
    {word:'If',count:1}
 ]

console.log(tenMostFrequentWords(paragraph, 10))
```

```
[{word:'love', count:6},
{word:'you', count:5},
{word:'can', count:3},
{word:'what', count:2},
{word:'teaching', count:2},
{word:'not', count:2},
{word:'else', count:2},
{word:'do', count:2},
{word:'I', count:2},
{word:'which', count:1}
]
```

Exercises: Level 3

1. Write a function which cleans text. Clean the following text. After cleaning, count three most frequent words in the string.

```
  sentence = `%I $am@% a %tea@cher%, &and& I lo%#ve
%tea@ching%;. There $is nothing; &as& mo@re rewarding as
educa@ting &and& @emp%o@wering peo@ple. ;I found
tea@ching m%o@re interesting tha@n any other %jo@bs.
%Do@es thi%s mo@tivate yo@u to be a tea@cher!?`
  console.log(cleanText(sentence))
```

```
 I am a teacher and I love teaching There is nothing as
more rewarding as educating and empowering people I found
teaching more interesting than any other jobs Does this
motivate you to be a teacher
 ```
```
```

```
    2. Write a function which find the most frequent words. After cleaning, count three most frequent words in the string.

    ```js
 console.log(mostFrequentWords(cleanedText))
  [{word:'I', count:3}, {word:'teaching', count:2}, {word:'teacher', count:2}]
```

🎉 CONGRATULATIONS ! DAY 12 COMPLETED 🎉
```

DAY-13 CONSOLE OBJECT METHODS

Console Object Methods

In this section, we will cover about console and console object methods. Absolute beginners usually do not know which to use: console.log(), document.write() or document.getElementById.

We use console object methods to show output on the browser console and we use document.write to show output on the browser document(view port). Both methods used only for testing and debugging purposes. The console method is the most popular testing and debugging tool on the browser. We use document.getElementById() when we like to interact with DOM try using JavaScript. We will cover DOM in another section.

In addition to the famous, console.log() method, the console provides other more methods.

console.log()

We use console.log() to show output on the browser console. We can substitute values and also we can style the logging out put using %c.

- Showing output on browser console

```
console.log('30 Days of JavaScript')

30 Days of JavaScript
```

- Substitution

```
console.log('%d %s of JavaScript', 30, 'Days')

30 Days of JavaScript
```

- CSS

We can style logging message using css. Copy the following code and paste it on browser console to see the result.

```
console.log('%c30 Days Of JavaScript', 'color:green') // log
output is green
console.log(
  '%c30 Days%c %cOf%c %cJavaScript%c',
  'color:green',
```

```
  '',
  'color:red',
  '',
  'color:yellow'
) // log output green red and yellow text
```

console.warn()

We use console.warn() to give warning on browser. For instance to inform or warn deprecation of version of a package or bad practices. Copy the following code and paste it on browser console to see warning messages.

```
console.warn('This is a warning')
console.warn(
  'You are using React. Do not touch the DOM. Virtual DOM will take care of handling the DOM!'
)
console.warn('Warning is different from error')
```

console.error()

The console.error() method shows an error messages.

```
console.error('This is an error message')
console.error('We all make mistakes')
```

console.table()

The console.table() method display data as a table on the console. Displays tabular data as a table. The console.table() takes one required argument data, which must be an array or an object, and one additional optional parameter columns.

Let us first start with a simple array. The code below displays a table with two columns. An index column to display the index and value column to display the names

```
const names = ['Asabeneh', 'Brook', 'David', 'John']
console.table(names)
```

Let us also check the result of an object. This creates table with two columns:an index column containing the keys and a value column contain the values of the object.

```
const user = {
  name: 'Asabeneh',
  title: 'Programmer',
  country: 'Finland',
  city: 'Helsinki',
  age: 250
}
console.table(user)
```

Check the rest of the examples by copying and paste on the browser console.

```
const countries = [
  ['Finland', 'Helsinki'],
  ['Sweden', 'Stockholm'],
  ['Norway', 'Oslo']
]
console.table(countries)
```

```
const users = [
  {
    name: 'Asabeneh',
    title: 'Programmer',
    country: 'Finland',
    city: 'Helsinki',
    age: 250
  },
  {
    name: 'Eyob',
    title: 'Teacher',
    country: 'Sweden',
    city: 'London',
    age: 25
  },
  {
    name: 'Asab',
    title: 'Instructor',
    country: 'Norway',
    city: 'Oslo',
    age: 22
  },
  {
    name: 'Matias',
    title: 'Developer',
    country: 'Denmark',
    city: 'Copenhagen',
```

```
    age: 28
  }
]
console.table(users)
```

console.time()

Starts a timer you can use to track how long an operation takes. You give each timer a unique name, and may have up to 10,000 timers running on a given page. When you call console.timeEnd() with the same name, the browser will output the time, in milliseconds, that elapsed since the timer was started.

```
const countries = [
  ['Finland', 'Helsinki'],
  ['Sweden', 'Stockholm'],
  ['Norway', 'Oslo']
]

console.time('Regular for loop')
for (let i = 0; i < countries.length; i++) {
  console.log(countries[i][0], countries[i][1])
}
console.timeEnd('Regular for loop')

console.time('for of loop')
for (const [name, city] of countries) {
  console.log(name, city)
}
console.timeEnd('for of loop')

console.time('forEach loop')
countries.forEach(([name, city]) => {
  console.log(name, city)
})
console.timeEnd('forEach loop')
```

```
Finland Helsinki
Sweden Stockholm
Norway Oslo
Regular for loop: 0.34716796875ms
Finland Helsinki
Sweden Stockholm
Norway Oslo
for of loop: 0.26806640625ms
Finland Helsinki
```

```
Sweden Stockholm
Norway Oslo
forEach loop: 0.358154296875ms
```

According the above output the regular for loop is slower than for of or forEach loop.

console.info()

It displays information message on browser console.

```
console.info('30 Days Of JavaScript challenge is trending on
Github')
console.info('30 Days Of fullStack challenge might be
released')
console.info('30 Days Of HTML and CSS challenge might be
released')
```

console.assert()

The console.assert() methods writes an error message to the console if the assertion is false. If the assertion is true, nothing happens. The first parameter is an assertion expression. If this expression is false, an Assertion failed error message will be displayed.

```
console.assert(4 > 3, '4 is greater than 3') // no result
console.assert(3 > 4, '3 is not greater than 4') // Assertion
failed: 3 is not greater than 4

for (let i = 0; i <= 10; i += 1) {
  let errorMessage = `${i} is not even`
  console.log('the # is ' + i)
  console.assert(i % 2 === 0, { number: i, errorMessage:
errorMessage })
}
```

console.group()

The console.group() can help to group different log groups. Copy the following code and paste it on browser console to the groups.

```
const names = ['Asabeneh', 'Brook', 'David', 'John']
const countries = [
  ['Finland', 'Helsinki'],
  ['Sweden', 'Stockholm'],
  ['Norway', 'Oslo']
]
const user = {
  name: 'Asabeneh',
  title: 'Programmer',
  country: 'Finland',
  city: 'Helsinki',
  age: 250
}
const users = [
  {
    name: 'Asabeneh',
    title: 'Programmer',
    country: 'Finland',
    city: 'Helsinki',
    age: 250
  },
  {
    name: 'Eyob',
    title: 'Teacher',
    country: 'Sweden',
    city: 'London',
    age: 25
  },
  {
    name: 'Asab',
    title: 'Instructor',
    country: 'Norway',
    city: 'Oslo',
    age: 22
  },
  {
    name: 'Matias',
    title: 'Developer',
    country: 'Denmark',
    city: 'Copenhagen',
    age: 28
  }
]

console.group('Names')
```

```
console.log(names)
console.groupEnd()

console.group('Countries')
console.log(countries)
console.groupEnd()

console.group('Users')
console.log(user)
console.log(users)
console.groupEnd()
```

console.count()

It prints the number of times the console.count() is called. It takes a string label parameter. It is very helpful to count the number of times a function is called. In the following example, the console.count() method will run three times

```
const func = () => {
  console.count('Function has been called')
}
func()
func()
func()
```

```
Function has been called: 1
Function has been called: 2
Function has been called: 3
```

console.clear()

The console.clear() cleans the browser console.

🍪 Keep up the good work. Keep pushing, the sky is the limit! You have just completed day 13 challenges and you are 13 steps a head in to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises:Level 1

1. Display the countries array as a table
2. Display the countries object as a table
3. Use console.group() to group logs

Exercises:Level 2

1. 10 > 2 * 10 use console.assert()
2. Write a warning message using console.warn()
3. Write an error message using console.error()

Exercises:Level 3

1. Check the speed difference among the following loops: while, for, for of, forEach

 CONGRATULATIONS ! 

DAY-14 ERROR HANDLING

JavaScript is a loosely-typed language. Some times you will get a runtime error when you try to access an undefined variable or call undefined function etc.

JavaScript similar to python or Java provides an error-handling mechanism to catch runtime errors using try-catch-finally block.

```
try {
  // code that may throw an error
} catch (err) {
  // code to be executed if an error occurs
} finally {
  // code to be executed regardless of an error occurs or not
}
```

try: wrap suspicious code that may throw an error in try block.The try statement allows us to define a block of code to be tested for errors while it is being executed.

catch: write code to do something in catch block when an error occurs. The catch block can have parameters that will give you error information. Catch block is used to log an error or display specific messages to the user.

finally: finally block will always be executed regardless of the occurrence of an error. The finally block can be used to complete the remaining task or reset variables that might have changed before error occurred in try block.

Example:

```
try {
  let lastName = 'Yetayeh'
  let fullName = fistName + ' ' + lastName
} catch (err) {
  console.log(err)
}
```

```
ReferenceError: fistName is not defined
    at <anonymous>:4:20
```

```
try {
  let lastName = 'Yetayeh'
  let fullName = fistName + ' ' + lastName
} catch (err) {
  console.error(err) // we can use console.log() or
console.error()
} finally {
  console.log('In any case I will be executed')
}
```

```
ReferenceError: fistName is not defined
    at <anonymous>:4:20
In any case it  will be executed
```

The catch block take a parameter. It is common to pass e, err or error as a parameter to the catch block. This parameter is an object and it has name and message keys. Lets use the name and message.

```
try {
  let lastName = 'Yetayeh'
  let fullName = fistName + ' ' + lastName
} catch (err) {
  console.log('Name of the error', err.name)
  console.log('Error message', err.message)
} finally {
  console.log('In any case I will be executed')
}
```

```
Name of the error ReferenceError
Error message fistName is not defined
In any case I will be executed
```

throw: the throw statement allows us to create a custom error. We can through a string, number, boolean or an object. Use the throw statement to throw an exception. When you throw an exception, expression specifies the value of the exception. Each of the following throws an exception:

```
throw 'Error2' // generates an exception with a string value
throw 42 // generates an exception with the value 42
throw true // generates an exception with the value true
throw new Error('Required') // generates an error object with
the message of Required
```

```
const throwErrorExampleFun = () => {
  let message
```

```
  let x = prompt('Enter a number: ')
  try {
    if (x == '') throw 'empty'
    if (isNaN(x)) throw 'not a number'
    x = Number(x)
    if (x < 5) throw 'too low'
    if (x > 10) throw 'too high'
  } catch (err) {
    console.log(err)
  }
}
throwErrorExampleFun()
```

Error Types

- ReferenceError: An illegal reference has occurred. A ReferenceError is thrown if we use a variable that has not been declared.

```
let firstName = 'Asabeneh'
let fullName = firstName + ' ' + lastName
console.log(fullName)
```

```
Uncaught ReferenceError: lastName is not defined
    at <anonymous>:2:35
```

- SyntaxError: A syntax error has occurred

```
let square = 2 x 2
console.log(square)
console.log('Hello, world")
```

```
Uncaught SyntaxError: Unexpected identifier
```

- TypeError: A type error has occurred

```
let num = 10
console.log(num.toLowerCase())
```

```
Uncaught TypeError: num.toLowerCase is not a function
    at <anonymous>:2:17
```

These are some of the common error you may face when you write a code. Understanding errors can help you to know what mistakes you made and it will help you to debug your code fast. 🍪 You are flawless. Now, you knew how to handle errors and you can write robust application which handle unexpected user inputs. You have just completed day 14 challenges and you are 14 steps a head in

to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Basic Error Handling

1. Write a try-catch block to catch an error when trying to access an undefined variable.
2. Create a function that throws an error if a number is negative.
3. Use finally in a try-catch block to print "Execution completed" regardless of an error.
4. Create a divide function that throws an error if dividing by zero

🎉 CONGRATULATIONS ! COMPLETED DAY-14

DAY-15 CLASSES

Classes

JavaScript is an object oriented programming language. Everything in JavScript is an object, with its properties and methods. We create class to create an object. A Class is like an object constructor, or a "blueprint" for creating objects. We instantiate a class to create an object. The class defines attributes and the behavior of the object, while the object, on the other hand, represents the class.

Once we create a class we can create object from it whenever we want. Creating an object from a class is called class instantiation.

In the object section, we saw how to create an object literal. Object literal is a singleton. If we want to get a similar object , we have to write it. However, class allows to create many objects. This helps to reduce amount of code and repetition of code.

Defining a classes

To define a class in JavaScript we need the keyword *class* , the name of a class in **CamelCase** and block code(two curly brackets). Let us create a class name Person.

```
// syntax
class ClassName {
    //  code goes here
}
```

Example:

```
class Person {
  // code goes here
}
```

We have created an Person class but it does not have any thing inside.

Class Instantiation

Instantiation class means creating an object from a class. We need the keyword *new* and we call the name of the class after the word new.

Let us create a dog object from our Person class.

```
class Person {
  // code goes here
}
const person = new Person()
console.log(person)
```

```
Person {}
```

As you can see, we have created a person object. Since the class did not have any properties yet the object is also empty.

Let use the class constructor to pass different properties for the class.

Class Constructor

The constructor is a builtin function which allows as to create a blueprint for our object. The constructor function starts with a keyword constructor followed by a parenthesis. Inside the parenthesis we pass the properties of the object as parameter. We use the *this* keyword to attach the constructor parameters with the class.

The following Person class constructor has firstName and lastName property. These properties are attached to the Person class using *this* keyword. *This* refers to the class itself.

```
class Person {
  constructor(firstName, lastName) {
    console.log(this) // Check the output from here
    this.firstName = firstName
    this.lastName = lastName
  }
}

const person = new Person()

console.log(person)
```

```
Person {firstName: undefined, lastName:undefined}
```

All the keys of the object are undefined. When ever we instantiate we should pass the value of the properties. Let us pass value at this time when we instantiate the class.

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh')

console.log(person1)
```

```
Person {firstName: "Asabeneh", lastName: "Yetayeh"}
```

As we have stated at the very beginning that once we create a class we can create many object using the class. Now, let us create many person objects using the Person class.

```
class Person {
  constructor(firstName, lastName) {
    console.log(this) // Check the output from here
    this.firstName = firstName
    this.lastName = lastName
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh')
const person2 = new Person('Lidiya', 'Tekle')
const person3 = new Person('Abraham', 'Yetayeh')

console.log(person1)
console.log(person2)
console.log(person3)
```

```
Person {firstName: "Asabeneh", lastName: "Yetayeh"}
Person {firstName: "Lidiya", lastName: "Tekle"}
Person {firstName: "Abraham", lastName: "Yetayeh"}
```

Using the class Person we created three persons object. As you can see our class did not many properties let us add more properties to the class.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    console.log(this) // Check the output from here
    this.firstName = firstName
```

```
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')

console.log(person1)
```

```
Person {firstName: "Asabeneh", lastName: "Yetayeh", age: 250,
country: "Finland", city: "Helsinki"}
```

Default values with constructor

The constructor function properties may have a default value like other regular functions.

```
class Person {
  constructor(
    firstName = 'Asabeneh',
    lastName = 'Yetayeh',
    age = 250,
    country = 'Finland',
    city = 'Helsinki'
  ) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
}

const person1 = new Person() // it will take the default
values
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')

console.log(person1)
console.log(person2)
```

```
Person {firstName: "Asabeneh", lastName: "Yetayeh", age: 250,
country: "Finland", city: "Helsinki"}
```

```
Person {firstName: "Lidiya", lastName: "Tekle", age: 28,
country: "Finland", city: "Espoo"}
```

Class methods

The constructor inside a class is a builtin function which allow us to create a blueprint for the object. In a class we can create class methods. Methods are JavaScript functions inside the class. Let us create some class methods.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')

console.log(person1.getFullName())
console.log(person2.getFullName())
```

```
Asabeneh Yetayeh
test.js:19 Lidiya Tekle
```

Properties with initial value

When we create a class for some properties we may have an initial value. For instance if you are playing a game, you starting score will be zero. So, we may have a starting score or score which is zero. In other way, we may have an initial skill and we will acquire some skill after some time.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
```

```
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')

console.log(person1.score)
console.log(person2.score)

console.log(person1.skills)
console.log(person2.skills)
```

```
0
0
[]
[]
```

A method could be regular method or a getter or a setter. Let us see, getter and setter.

getter

The get method allow us to access value from the object. We write a get method using keyword *get* followed by a function. Instead of accessing properties directly from the object we use getter to get the value. See the example bellow

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
```

```
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')

console.log(person1.getScore) // We do not need parenthesis to
call a getter method
console.log(person2.getScore)

console.log(person1.getSkills)
console.log(person2.getSkills)
```

```
0
0
[]
[]
```

setter

The setter method allow us to modify the value of certain properties. We write a setter method using keyword *set* followed by a function. See the example bellow.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
```

```
  get getSkills() {
    return this.skills
  }
  set setScore(score) {
    this.score += score
  }
  set setSkill(skill) {
    this.skills.push(skill)
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')

person1.setScore = 1
person1.setSkill = 'HTML'
person1.setSkill = 'CSS'
person1.setSkill = 'JavaScript'

person2.setScore = 1
person2.setSkill = 'Planning'
person2.setSkill = 'Managing'
person2.setSkill = 'Organizing'

console.log(person1.score)
console.log(person2.score)

console.log(person1.skills)
console.log(person2.skills)
```

```
1
1
["HTML", "CSS", "JavaScript"]
["Planning", "Managing", "Organizing"]
```

Do not be puzzled by the difference between regular method and a getter. If you know how to make a regular method you are good. Let us add regular method called getPersonInfo in the Person class.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
```

```
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
  set setScore(score) {
    this.score += score
  }
  set setSkill(skill) {
    this.skills.push(skill)
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ')
+
        ` and ${this.skills[this.skills.length - 1]}`
    let formattedSkills = skills ? `He knows ${skills}` : ''

    let info = `${fullName} is ${this.age}. He lives
${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
}

const person1 = new Person('Asabeneh', 'Yetayeh', 250,
'Finland', 'Helsinki')
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland',
'Espoo')
const person3 = new Person('John', 'Doe', 50, 'Mars', 'Mars
city')

person1.setScore = 1
person1.setSkill = 'HTML'
person1.setSkill = 'CSS'
person1.setSkill = 'JavaScript'

person2.setScore = 1
person2.setSkill = 'Planning'
person2.setSkill = 'Managing'
person2.setSkill = 'Organizing'

console.log(person1.getScore)
```

```
console.log(person2.getScore)

console.log(person1.getSkills)
console.log(person2.getSkills)
console.log(person3.getSkills)

console.log(person1.getPersonInfo())
console.log(person2.getPersonInfo())
console.log(person3.getPersonInfo())
```

```
1
1
["HTML", "CSS", "JavaScript"]
["Planning", "Managing", "Organizing"]
[]
Asabeneh Yetayeh is 250. He lives Helsinki, Finland. He knows
HTML, CSS and JavaScript
Lidiya Tekle is 28. He lives Espoo, Finland. He knows
Planning, Managing and Organizing
John Doe is 50. He lives Mars city, Mars.
```

Static method

The static keyword defines a static method for a class. Static methods are not called on instances of the class. Instead, they are called on the class itself. These are often utility functions, such as functions to create or clone objects. An example of static method is *Date.now()*. The *now* method is called directly from the class.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
```

```
  set setScore(score) {
    this.score += score
  }
  set setSkill(skill) {
    this.skills.push(skill)
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ')
+
        ` and ${this.skills[this.skills.length - 1]}`

    let formattedSkills = skills ? `He knows ${skills}` : ''

    let info = `${fullName} is ${this.age}. He lives
${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
  static favoriteSkill() {
    const skills = ['HTML', 'CSS', 'JS', 'React', 'Python',
'Node']
    const index = Math.floor(Math.random() * skills.length)
    return skills[index]
  }
  static showDateTime() {
    let now = new Date()
    let year = now.getFullYear()
    let month = now.getMonth() + 1
    let date = now.getDate()
    let hours = now.getHours()
    let minutes = now.getMinutes()
    if (hours < 10) {
      hours = '0' + hours
    }
    if (minutes < 10) {
      minutes = '0' + minutes
    }

    let dateMonthYear = date + '.' + month + '.' + year
    let time = hours + ':' + minutes
    let fullTime = dateMonthYear + ' ' + time
    return fullTime
  }
}

console.log(Person.favoriteSkill())
console.log(Person.showDateTime())
```

```
Node
15.1.2020 23:56
```

The static methods are methods which can be used as utility functions.

Inheritance

Using inheritance we can access all the properties and the methods of the parent class. This reduces repetition of code. If you remember, we have a Person parent class and we will create children from it. Our children class could be student, teach etc.

```
// syntax
class ChildClassName extends {
 // code goes here
}
```

Let us create a Student child class from Person parent class.

```
class Student extends Person {
  saySomething() {
    console.log('I am a child of the person class')
  }
}

const s1 = new Student('Asabeneh', 'Yetayeh', 'Finland', 250,
'Helsinki')
console.log(s1)
console.log(s1.saySomething())
console.log(s1.getFullName())
console.log(s1.getPersonInfo())
```

```
Student {firstName: "Asabeneh", lastName: "Yetayeh", age:
"Finland", country: 250, city: "Helsinki", …}
I am a child of the person class
Asabeneh Yetayeh
Student {firstName: "Asabeneh", lastName: "Yetayeh", age:
"Finland", country: 250, city: "Helsinki", …}
Asabeneh Yetayeh is Finland. He lives Helsinki, 250.
```

Overriding methods

As you can see, we manage to access all the methods in the Person Class and we used it in the Student child class. We can customize the parent methods, we can add additional properties to a child class. If we want to customize, the methods and if we want to add extra properties, we need to use the constructor function the child class too. Inside the constructor function we call the super() function to access all the properties from the parent class. The Person class didn't have gender but now let us give gender property for the child class, Student. If the same method name used in the child class, the parent method will be overridden.

```
class Student extends Person {
  constructor(firstName, lastName, age, country, city, gender)
{
    super(firstName, lastName, age, country, city)
    this.gender = gender
  }

  saySomething() {
    console.log('I am a child of the person class')
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ')
+
        ` and ${this.skills[this.skills.length - 1]}`

    let formattedSkills = skills ? `He knows ${skills}` : ''
    let pronoun = this.gender == 'Male' ? 'He' : 'She'

    let info = `${fullName} is ${this.age}. ${pronoun} lives
in ${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
}

const s1 = new Student(
  'Asabeneh',
  'Yetayeh',
  250,
  'Finland',
  'Helsinki',
  'Male'
)
```

```
const s2 = new Student('Lidiya', 'Tekle', 28, 'Finland',
'Helsinki', 'Female')
s1.setScore = 1
s1.setSkill = 'HTML'
s1.setSkill = 'CSS'
s1.setSkill = 'JavaScript'

s2.setScore = 1
s2.setSkill = 'Planning'
s2.setSkill = 'Managing'
s2.setSkill = 'Organizing'

console.log(s1)

console.log(s1.saySomething())
console.log(s1.getFullName())
console.log(s1.getPersonInfo())

console.log(s2.saySomething())
console.log(s2.getFullName())
console.log(s2.getPersonInfo())
```

```
Student {firstName: "Asabeneh", lastName: "Yetayeh", age: 250,
country: "Finland", city: "Helsinki", …}
Student {firstName: "Lidiya", lastName: "Tekle", age: 28,
country: "Finland", city: "Helsinki", …}
I am a child of the person class
Asabeneh Yetayeh
Student {firstName: "Asabeneh", lastName: "Yetayeh", age: 250,
country: "Finland", city: "Helsinki", …}
Asabeneh Yetayeh is 250. He lives in Helsinki, Finland. He
knows HTML, CSS and JavaScript
I am a child of the person class
Lidiya Tekle
Student {firstName: "Lidiya", lastName: "Tekle", age: 28,
country: "Finland", city: "Helsinki", …}
Lidiya Tekle is 28. She lives in Helsinki, Finland. He knows
Planning, Managing and Organizing
```

Now, the getPersonInfo method has been overridden and it identifies if the person is male or female.

🍪 You are excelling. Now, you knew class and you have the power to turn everything to an object. You made it to half way to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises Level 1

1. Create an Animal class. The class will have name, age, color, legs properties and create different methods
2. Create a Dog and Cat child class from the Animal Class.

Exercises Level 2

1. Override the method you create in Animal class

Exercises Level 3

1. Let's try to develop a program which calculate measure of central tendency of a sample(mean, median, mode) and measure of variability(range, variance, standard deviation). In addition to those measures find the min, max, count, percentile, and frequency distribution of the sample. You can create a class called Statistics and create all the functions which do statistical calculations as method for the Statistics class. Check the output below.

```
ages = [31, 26, 34, 37, 27, 26, 32, 32, 26, 27, 27, 24, 32,
33, 27, 25, 26, 38, 37, 31, 34, 24, 33, 29, 26]

console.log('Count:', statistics.count()) // 25
console.log('Sum: ', statistics.sum()) // 744
console.log('Min: ', statistics.min()) // 24
console.log('Max: ', statistics.max()) // 38
console.log('Range: ', statistics.range() // 14
console.log('Mean: ', statistics.mean()) // 30
console.log('Median: ',statistics.median()) // 29
console.log('Mode: ', statistics.mode()) // {'mode': 26,
'count': 5}
console.log('Variance: ',statistics.var()) // 17.5
console.log('Standard Deviation: ', statistics.std()) // 4.2
console.log('Variance: ',statistics.var()) // 17.5
console.log('Frequency Distribution: ',statistics.freqDist())
// [(20.0, 26), (16.0, 27), (12.0, 32), (8.0, 37), (8.0, 34),
(8.0, 33), (8.0, 31), (8.0, 24), (4.0, 38), (4.0, 29), (4.0,
25)]
```

```
// you output should look like this
console.log(statistics.describe())
Count: 25
Sum:  744
Min:  24
Max:  38
Range:  14
Mean:  30
Median:  29
Mode:  (26, 5)
Variance:  17.5
Standard Deviation:  4.2
Frequency Distribution: [(20.0, 26), (16.0, 27), (12.0, 32),
(8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31), (8.0, 24), (4.0,
38), (4.0, 29), (4.0, 25)]
```

2. Create a class called PersonAccount. It has firstname, lastname, incomes, expenses properties and it has totalIncome, totalExpense, accountInfo,addIncome, addExpense and accountBalance methods. Incomes is a set of incomes and its description and expenses is also a set of expenses and its description.

🎉 CONGRATULATIONS ! 15 DAYS COMPLETED 🎉

DAY-16 JSON

JSON stands for JavaScript Object Notation. The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text or string only. JSON is a light weight data format for storing and transporting. JSON is mostly used when data is sent from a server to a client. JSON is an easier-to-use alternative to XML.

Example:

```
{
"users":[
  {
    "firstName":"Asabeneh",
    "lastName":"Yetayeh",
    "age":250,
    "email":"asab@asb.com"
  },
  {
    "firstName":"Alex",
    "lastName":"James",
    "age":25,
    "email":"alex@alex.com"
  },
  {
  "firstName":"Lidiya",
  "lastName":"Tekle",
  "age":28,
  "email":"lidiya@lidiya.com"
  }
]
}
```

The above JSON example is not much different from a normal object. Then, what is the difference ? The difference is the key of a JSON object should be with double quotes or it should be a string. JavaScript Object and JSON are very similar that we can change JSON to Object and Object to JSON.

Let us see the above example in more detail, it starts with a curly bracket. Inside the curly bracket, there is "users" key which has a value array. Inside the array we have different objects and each objects has keys, each keys has to have double quotes. For instance, we use "firstNaMe" instead of just firstName, however in

object we use keys without double quotes. This is the major difference between an object and a JSON. Let's see more examples about JSON.

Example:

```
{
    "Alex": {
        "email": "alex@alex.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 30
    },
    "Asab": {
        "email": "asab@asab.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "Redux",
            "MongoDB",
            "Express",
            "React",
            "Node"
        ],
        "age": 25,
        "isLoggedIn": false,
        "points": 50
    },
    "Brook": {
        "email": "daniel@daniel.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React",
            "Redux"
        ],
        "age": 30,
        "isLoggedIn": true,
        "points": 50
    },
    "Daniel": {
        "email": "daniel@alex.com",
        "skills": [
            "HTML",
            "CSS",
```

```
            "JavaScript",
            "Python"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    },
    "John": {
        "email": "john@john.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React",
            "Redux",
            "Node.js"
        ],
        "age": 20,
        "isLoggedIn": true,
        "points": 50
    },
    "Thomas": {
        "email": "thomas@thomas.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    },
    "Paul": {
        "email": "paul@paul.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "MongoDB",
            "Express",
            "React",
            "Node"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    }
}
```

Converting JSON to JavaScript Object

Mostly we fetch JSON data from HTTP response or from a file, but we can store the JSON as a string and we can change to Object for sake of demonstration. In JavaScript the keyword *JSON* has *parse()* and *stringify()* methods. When we want to change the JSON to an object we parse the JSON using *JSON.parse()*. When we want to change the object to JSON we use *JSON.stringify()*.

JSON.parse()

```
JSON.parse(json[, reviver])
// json or text , the data
// reviver is an optional callback function
/* JSON.parse(json, (key, value) => {

})
*/
```

```
const usersText = `{
"users":[
  {
    "firstName":"Asabeneh",
    "lastName":"Yetayeh",
    "age":250,
    "email":"asab@asb.com"
  },
  {
    "firstName":"Alex",
    "lastName":"James",
    "age":25,
    "email":"alex@alex.com"
  },
  {
  "firstName":"Lidiya",
  "lastName":"Tekle",
  "age":28,
  "email":"lidiya@lidiya.com"
  }
]
}`

const usersObj = JSON.parse(usersText, undefined, 4)
console.log(usersObj)
```

Using a reviver function with JSON.parse()

To use the reviver function as a formatter, we put the keys we want to format first name and last name value. Let us say, we are interested to format the firstName and lastName of the JSON data .

```
const usersText = `{
"users":[
  {
    "firstName":"Asabeneh",
    "lastName":"Yetayeh",
    "age":250,
    "email":"asab@asb.com"
  },
  {
    "firstName":"Alex",
    "lastName":"James",
    "age":25,
    "email":"alex@alex.com"
  },
  {
  "firstName":"Lidiya",
  "lastName":"Tekle",
  "age":28,
  "email":"lidiya@lidiya.com"
  }
]
}`

const usersObj = JSON.parse(usersText, (key, value) => {
  let newValue =
    typeof value == 'string' && key != 'email' ?
value.toUpperCase() : value
  return newValue
})
console.log(usersObj)
```

The *JSON.parse()* is very handy to use. You do not have to pass optional parameter, you can just use it with the required parameter and you will achieve quite a lot.

Converting Object to JSON

When we want to change the object to JSON we use *JSON.stringify()*. The stringify method takes one required parameter and two optional parameters. The replacer is used as filter and the space is an indentations. If we do not want to filter out any of the keys from the object we can just pass undefined.

```
JSON.stringify(obj, replacer, space)
// json or text , the data
// reviver is an optional callback function
```

Let us convert the following object to a string. First let use keep all the keys and also let us have 4 space indentation.

```
const users = {
  Alex: {
    email: 'alex@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript'],
    age: 20,
    isLoggedIn: false,
    points: 30
  },
  Asab: {
    email: 'asab@asab.com',
    skills: [
      'HTML',
      'CSS',
      'JavaScript',
      'Redux',
      'MongoDB',
      'Express',
      'React',
      'Node'
    ],
    age: 25,
    isLoggedIn: false,
    points: 50
  },
  Brook: {
    email: 'daniel@daniel.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux'],
    age: 30,
    isLoggedIn: true,
    points: 50
  },
  Daniel: {
    email: 'daniel@alex.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'Python'],
    age: 20,
    isLoggedIn: false,
```

```
    points: 40
  },
  John: {
    email: 'john@john.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux',
'Node.js'],
    age: 20,
    isLoggedIn: true,
    points: 50
  },
  Thomas: {
    email: 'thomas@thomas.com',
    skills: ['HTML', 'CSS', 'JavaScript', 'React'],
    age: 20,
    isLoggedIn: false,
    points: 40
  },
  Paul: {
    email: 'paul@paul.com',
    skills: [
      'HTML',
      'CSS',
      'JavaScript',
      'MongoDB',
      'Express',
      'React',
      'Node'
    ],
    age: 20,
    isLoggedIn: false,
    points: 40
  }
}

const txt = JSON.stringify(users, undefined, 4)
console.log(txt) // text means JSON- because json is a string
form of an object.
```

```
{
    "Alex": {
    "email": "alex@alex.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript"
    ],
    "age": 20,
    "isLoggedIn": false,
```

```
    "points": 30
  },
  "Asab": {
    "email": "asab@asab.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "Redux",
      "MongoDB",
      "Express",
      "React",
      "Node"
    ],
    "age": 25,
    "isLoggedIn": false,
    "points": 50
  },
  "Brook": {
    "email": "daniel@daniel.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "React",
      "Redux"
    ],
    "age": 30,
    "isLoggedIn": true,
    "points": 50
  },
  "Daniel": {
    "email": "daniel@alex.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "Python"
    ],
    "age": 20,
    "isLoggedIn": false,
    "points": 40
  },
  "John": {
```

```
    "email": "john@john.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "React",
      "Redux",
      "Node.js"
    ],
    "age": 20,
    "isLoggedIn": true,
    "points": 50
  },
  "Thomas": {
    "email": "thomas@thomas.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "React"
    ],
    "age": 20,
    "isLoggedIn": false,
    "points": 40
  },
  "Paul": {
    "email": "paul@paul.com",
    "skills": [
      "HTML",
      "CSS",
      "JavaScript",
      "MongoDB",
      "Express",
      "React",
      "Node"
    ],
    "age": 20,
    "isLoggedIn": false,
    "points": 40
  }
}
```

Using a Filter Array with JSON.stringify

Now, lets use the replacer as a filter. The user object has long list of keys but we are interested only in few of them. We put the keys we want to keep in array as show in the example and use it the place of the replacer.

```
const user = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  country: 'Finland',
  city: 'Helsinki',
  email: 'alex@alex.com',
  skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Python'],
  age: 250,
  isLoggedIn: false,
  points: 30
}

const txt = JSON.stringify(user,['firstName', 'lastName',
'country', 'city', 'age'],4)
console.log(txt)
```

```
{
    "firstName": "Asabeneh",
    "lastName": "Yetayeh",
    "country": "Finland",
    "city": "Helsinki",
    "age": 250
}
```

🍪 You are extraordinary. Now, you knew a light-weight data format which you may use to store data or to send it an HTTP server. You are 16 steps a head to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

```
const skills = ['HTML', 'CSS', 'JS', 'React','Node', 'Python']
let age = 250;
let isMarried   = true
const student = {
  firstName:'Asabeneh',
  lastName:'Yetayehe',
  age:250,
  isMarried:true,
  skills:['HTML', 'CSS', 'JS', 'React','Node', 'Python', ]
}
const txt = `{
    "Alex": {
        "email": "alex@alex.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 30
    },
    "Asab": {
        "email": "asab@asab.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "Redux",
            "MongoDB",
            "Express",
            "React",
            "Node"
        ],
        "age": 25,
        "isLoggedIn": false,
        "points": 50
    },
    "Brook": {
        "email": "daniel@daniel.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React",
            "Redux"
        ],
        "age": 30,
        "isLoggedIn": true,
```

```
        "points": 50
    },
    "Daniel": {
        "email": "daniel@alex.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "Python"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    },
    "John": {
        "email": "john@john.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React",
            "Redux",
            "Node.js"
        ],
        "age": 20,
        "isLoggedIn": true,
        "points": 50
    },
    "Thomas": {
        "email": "thomas@thomas.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "React"
        ],
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    },
    "Paul": {
        "email": "paul@paul.com",
        "skills": [
            "HTML",
            "CSS",
            "JavaScript",
            "MongoDB",
            "Express",
            "React",
            "Node"
        ],
```

```
        "age": 20,
        "isLoggedIn": false,
        "points": 40
    }
}
`
```

Exercises Level 1

1. Change skills array to JSON using JSON.stringify()
2. Stringify the age variable
3. Stringify the isMarried variable
4. Stringify the student object

Exercises Level 2

1. Stringify the students object with only firstName, lastName and skills properties

Exercises Level 3

1. Parse the *txt* JSON to object.
2. Find the user who has many skills from the variable stored in *txt*.

🎉 CONGRATULATIONS ! DAY-16 COMPLETED 🎉

DAY-17 WEB PAGES

HTML5 Web Storage

Web Storage(sessionStorage and localStorage) is a new HTML5 API offering important benefits over traditional cookies. Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance. The data storage limit of cookies in many web browsers is about 4 KB per cookie. We Storages can store far larger data (at least 5MB) and never transferred to the server. All sites from the same or one origin can store and access the same data.

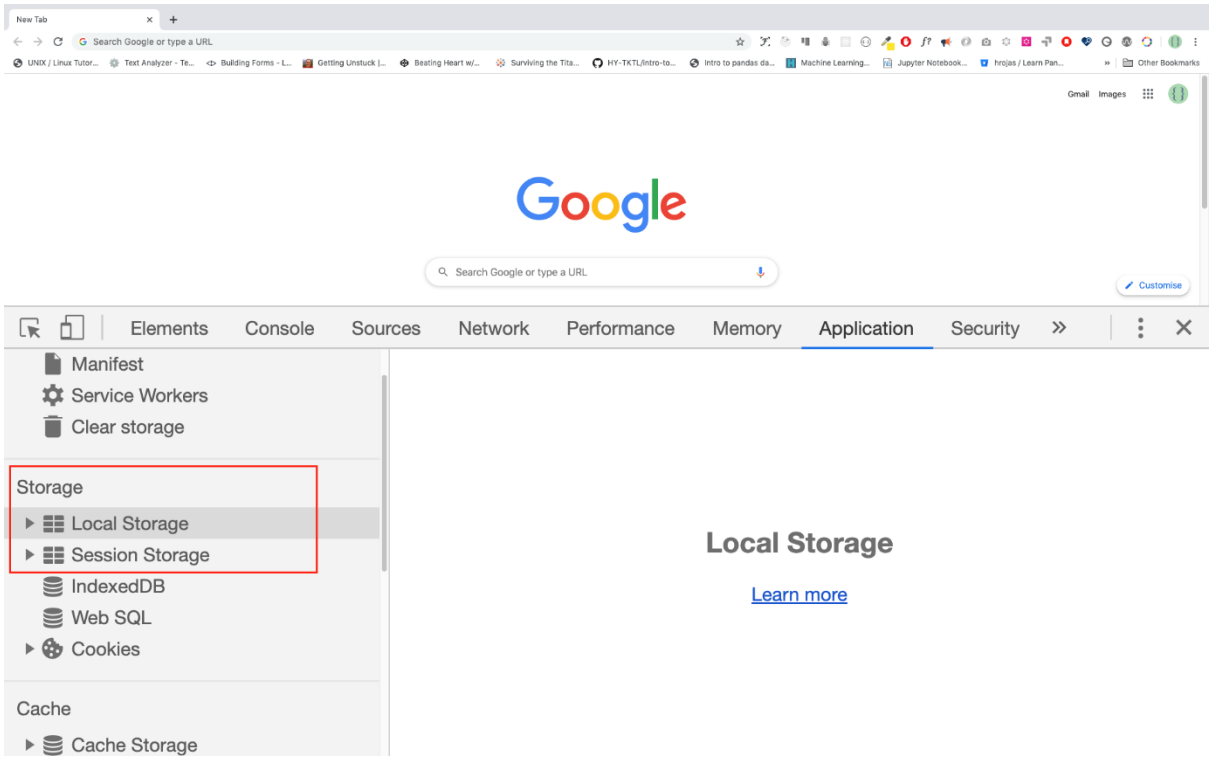
The data being stored can be accessed using JavaScript, which gives you the ability to leverage client-side scripting to do many things that have traditionally involved server-side programming and relational databases. There are two Web Storage objects:

- sessionStorage
- localStorage

localStorage is similar to sessionStorage, except that while data stored in localStorage has no expiration time, data stored in sessionStorage gets cleared when the page session ends — that is, when the page is closed.

It should be noted that data stored in either localStorage or sessionStorage is specific to the protocol of the page.

The keys and the values are always strings (note that, as with objects, integer keys will be automatically converted to strings).



sessionStorage

sessionStorage is only available within the browser tab or window session. It's designed to store data in a single web page session. That means if the window is closed the session data will be removed. Since sessionStorage and localStorage has similar methods, we will focus only on localStorage.

localStorage

The HTML5 localStorage is the para of the web storage API which is used to store data on the browser with no expiration data. The data will be available on the browser even after the browser is closed. localStorage is kept even between browser sessions. This means data is still available when the browser is closed and reopened, and also instantly between tabs and windows.

Web Storage data is, in both cases, not available between different browsers. For example, storage objects created in Firefox cannot be accessed in Internet Explorer, exactly like cookies. There are five methods to work on local storage: *setItem(), getItem(), removeItem(), clear(), key()*

Use case of Web Storages

Some use case of Web Storages are

- store data temporarily
- saving products that the user places in his shopping cart
- data can be made available between page requests, multiple browser tabs, and also between browser sessions using localStorage
- can be used offline completely using localStorage
- Web Storage can be a great performance win when some static data is stored on the client to minimize the number of subsequent requests. Even images can be stored in strings using Base64 encoding.
- can be used for user authentication method

For the examples mentioned above, it makes sense to use localStorage. You may be wondering, then, when we should use sessionStorage.

In cases, we want to to get rid of the data as soon as the window is closed. Or, perhaps, if we do not want the application to interfere with the same application that's open in another window. These scenarios are served best with sessionStorage.

Now, let us see how make use of these Web Storage APIs.

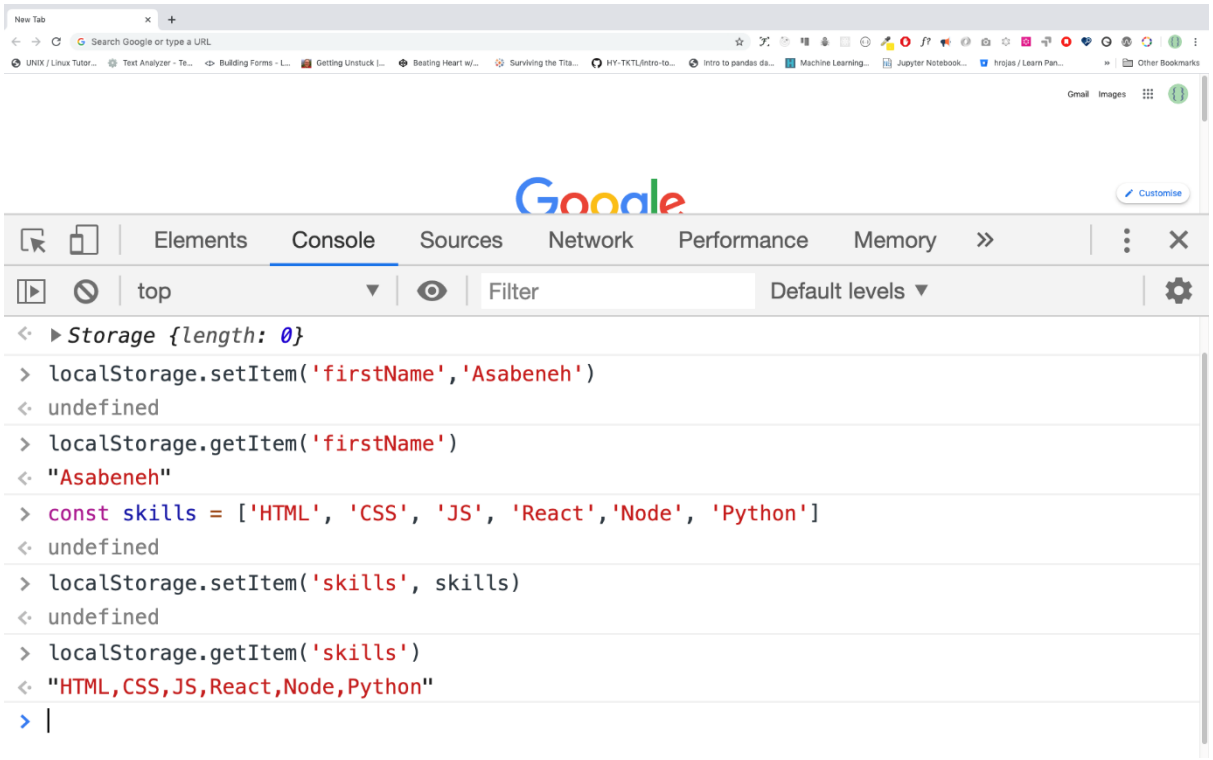
HTML5 Web Storage Objects

HTML web storage provides two objects for storing data on the client:

- window.localStorage - stores data with no expiration date
- window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)Most modern browsers support Web Storage, however it is good to check browser support for localStorage and sessionStorage. Let us see the available methods for the Web Storage objects.

Web Storage objects:

- *localStorage* - to display the localStorage object
- *localStorage.clear()* - to remove everything in the local storage
- *localStorage.setItem()* - to store data in the localStorage. It takes a key and a value parameters.
- *localStorage.getItem()* - to display data stored in the localStorage. It takes a key as a parameter.
- *localStorage.removeItem()* - to remove stored item form a localStorage. It takes key as a parameter.
- *localStorage.key()* - to display a data stored in a localStorage. It takes index as a parameter.



Setting item to the localStorage

When we set data to be stored in a localStorage, it will be stored as a string. If we are storing an array or an object, we should stringify it first to keep the format unless otherwise we lose the array structure or the object structure of the original data.

We store data in the localStorage using the *localStorage.setItem* method.

```
//syntax
localStorage.setItem('key', 'value')
```

- Storing string in a localStorage

```
localStorage.setItem('firstName', 'Asabeneh') // since the value is string we do not stringify it
console.log(localStorage)
```

```
Storage {firstName: 'Asabeneh', length: 1}
```

- Storing number in a local storage

```
localStorage.setItem('age', 200)
console.log(localStorage)
```

```
 Storage {age: '200', firstName: 'Asabeneh', length: 2}
```

- Storing an array in a localStorage. If we are storing an array, an object or object array, we should stringify the object first. See the example below.

```
const skills = ['HTML', 'CSS', 'JS', 'React']
//Skills array has to be stringified first to keep the format.
const skillsJSON = JSON.stringify(skills, undefined, 4)
localStorage.setItem('skills', skillsJSON)
console.log(localStorage)
```

```
Storage {age: '200', firstName: 'Asabeneh', skills:
'HTML,CSS,JS,React', length: 3}
```

```
let skills = [
  { tech: 'HTML', level: 10 },
  { tech: 'CSS', level: 9 },
  { tech: 'JS', level: 8 },
  { tech: 'React', level: 9 },
  { tech: 'Redux', level: 10 },
  { tech: 'Node', level: 8 },
  { tech: 'MongoDB', level: 8 }
]

let skillJSON = JSON.stringify(skills)
localStorage.setItem('skills', skillJSON)
```

- Storing an object in a localStorage. Before we storage objects to a localStorage, the object has to be stringified.

```
const user = {
  firstName: 'Asabeneh',
  age: 250,
  skills: ['HTML', 'CSS', 'JS', 'React']
}

const userText = JSON.stringify(user, undefined, 4)
localStorage.setItem('user', userText)
```

Getting item from localStorage

We get data from the local storage using *localStorage.getItem()* method.

```
//syntax
localStorage.getItem('key')
```

```
let firstName = localStorage.getItem('firstName')
let age = localStorage.getItem('age')
let skills = localStorage.getItem('skills')
console.log(firstName, age, skills)
```

```
 'Asabeneh', '200', '['HTML','CSS','JS','React']'
```

As you can see the skill is in a string format. Let us use JSON.parse() to parse it to normal array.

```
let skills = localStorage.getItem('skills')
let skillsObj = JSON.parse(skills, undefined, 4)
console.log(skillsObj)
```

```
['HTML','CSS','JS','React']
```

Clearing the localStorage

The clear method, will clear everything stored in the local storage

```
localStorage.clear()
```

 You are determined .Now, you knew a Web Storages and you knew how to store small data on client browsers. You are 17 steps a head to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises: Level 1

1. Store you first name, last name, age, country, city in your browser localStorage.

Exercises: Level 2

1. Create a student object. The student object will have first name, last name, age, skills, country, enrolled keys and values for the keys. Store the student object in your browser localStorage.

Exercises: Level 3

1. Create an object called personAccount. It has firstname, lastname, incomes, expenses properties and it has totalIncome, totalExpense, accountInfo,addIncome, addExpense and accountBalance methods. Incomes is a set of incomes and its description and expenses is also a set of expenses and its description.

 CONGRATULATIONS ! 

DAY-18 PROMISE

Promise

We human give or receive a promise to do some activity at some point in time. If we keep the promise we make others happy but if we do not keep the promise, it may lead discontentment. Promise in JavaScript has something in common with the above examples.

A Promise is a way to handle asynchronous operations in JavaScript. It allows handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

A Promise is in one of these states:

- pending: initial state, neither fulfilled nor rejected.
- fulfilled: meaning that the operation completed successfully.
- rejected: meaning that the operation failed.

A pending promise can either be fulfilled with a value, or rejected with a reason (error). When either of these options happens, the associated handlers queued up by a promise's then method are called. (If the promise has already been fulfilled or rejected when a corresponding handler is attached, the handler will be called, so there is no race condition between an asynchronous operation completing and its handlers being attached.)

As the Promise.prototype.then() and Promise.prototype.catch() methods return promises, they can be chained.

Callbacks

To understand promise very well let us understand callback first. Let's see the following callbacks. From the following code blocks you will notice, the difference between callback and promises.

- call back Let us see a callback function which can take two parameters. The first parameter is err and the second is result. If the err parameter is false, there will not be error other wise it will return an error.

In this case the err has a value and it will return the err block.

```
//Callback
const doSomething = callback => {
  setTimeout(() => {
    const skills = ['HTML', 'CSS', 'JS']
    callback('It did not go well', skills)
  }, 2000)
}

const callback = (err, result) => {
  if (err) {
    return console.log(err)
  }
  return console.log(result)
}

doSomething(callback)
```

```
// after 2 seconds it will print
It did not go well
```

In this case the err is false and it will return the else block which is the result.

```
const doSomething = callback => {
  setTimeout(() => {
    const skills = ['HTML', 'CSS', 'JS']
    callback(false, skills)
  }, 2000)
}

doSomething((err, result) => {
  if (err) {
    return console.log(err)
  }
  return console.log(result)
})
```

```
// after 2 seconds it will print the skills
["HTML", "CSS", "JS"]
```

Promise constructor

We can create a promise using the Promise constructor. We can create a new promise using the key word new followed by the word Promise and followed by a parenthesis. Inside the parenthesis, it takes a callback function. The promise callback function has two parameters which are the *resolve* and *reject* functions.

```
// syntax
const promise = new Promise((resolve, reject) => {
  resolve('success')
  reject('failure')
})
```

```
// Promise
const doPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const skills = ['HTML', 'CSS', 'JS']
    if (skills.length > 0) {
      resolve(skills)
    } else {
      reject('Something wrong has happened')
    }
  }, 2000)
})

doPromise
  .then(result => {
    console.log(result)
  })
  .catch(error => console.log(error))
```

```
["HTML", "CSS", "JS"]
```

The above promise has been settled with resolve. Let us another example when the promise is settled with reject.

```
// Promise
const doPromise = new Promise((resolve, reject) => {
  setTimeout(() => {
    const skills = ['HTML', 'CSS', 'JS']
    if (skills.includes('Node')) {
      resolve('fullstack developer')
```

```
    } else {
      reject('Something wrong has happened')
    }
  }, 2000)
})

doPromise
  .then(result => {
    console.log(result)
  })
  .catch(error => console.error(error))
```

Something wrong has happened

Fetch API

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used XMLHttpRequest, but the new API provides a more powerful and flexible feature set. In this challenge we will use fetch to request url and APIS. In addition to that let us see demonstrate use case of promises in accessing network resources using the fetch API.

```
const url = 'https://restcountries.com/v2/all' // countries
api
fetch(url)
  .then(response => response.json()) // accessing the API data
as JSON
  .then(data => {
    // getting the data
    console.log(data)
  })
  .catch(error => console.error(error)) // handling error if
something wrong happens
```

Async and Await

Async and await is an elegant way to handle promises. It is easy to understand and it clean to write.

```
const square = async function (n) {
  return n * n
}

square(2)

Promise {<resolved>: 4}
```

The word *async* in front of a function means that function will return a promise. The above square function instead of a value it returns a promise.

How do we access the value from the promise? To access the value from the promise, we will use the keyword *await*.

```
const square = async function (n) {
  return n * n
}
const value = await square(2)
console.log(value)

4
```

Now, as you can see from the above example writing async in front of a function create a promise and to get the value from a promise we use await. Async and await go together, one can not exist without the other.

Let us fetch API data using both promise method and async and await method.

- promise

```
const url = 'https://restcountries.com/v2/all'
fetch(url)
  .then(response => response.json())
  .then(data => {
    console.log(data)
  })
  .catch(error => console.error(error))
```

- async and await

```
const fetchData = async () => {
  try {
    const response = await fetch(url)
    const countries = await response.json()
    console.log(countries)
  } catch (err) {
    console.error(err)
  }
}
console.log('===== async and await')
fetchData()
```

🍪 You are real and you kept your promise and you reached to day 18. Keep your promise and settle the challenge with resolve. You are 18 steps ahead to your way to greatness. Now do some exercises for your brain and muscles.

Exercises

```
const countriesAPI = 'https://restcountries.com/v2/all'
const catsAPI = 'https://api.thecatapi.com/v1/breeds'
```

Exercises: Level 1

1. Read the countries API using fetch and print the name of country, capital, languages, population and area.

Exercises: Level 2

1. Print out all the cat names in to catNames variable.

Exercises: Level 3

1. Read the cats api and find the average weight of cat in metric unit.
2. Read the countries api and find out the 10 largest countries
3. Read the countries api and count total number of languages in the world used as officials.

🎉 CONGRATULATIONS ! DAY 18 COMPLETED 🎉

DAY-19 CLOSURES

Closure

JavaScript allows writing function inside an outer function. We can write as many inner functions as we want. If inner function access the variables of outer function then it is called closure.

```
function outerFunction() {
    let count = 0;
    function innerFunction() {
        count++
        return count
    }

    return innerFunction
}
const innerFunc = outerFunction()

console.log(innerFunc())
console.log(innerFunc())
console.log(innerFunc())
```

```
1
2
3
```

Let us more example of inner functions

```
function outerFunction() {
    let count = 0;
    function plusOne() {
        count++
        return count
    }
    function minusOne() {
        count--
        return count
    }

    return {
        plusOne:plusOne(),
        minusOne:minusOne()
    }
}
```

```
const innerFuncs = outerFunction()

console.log(innerFuncs.plusOne)
console.log(innerFuncs.minusOne)
```

```
1
0
```

 You are making progress. Maintain your momentum, keep the good work. Now do some exercises for your brain and for your muscle.

Exercises

Exercises: Level 1

1. Create a closure which has one inner function

Exercises: Level 2

1. Create a closure which has three inner functions

Exercises: Level 3

1. Create a personAccount out function. It has firstname, lastname, incomes, expenses inner variables. It has totalIncome, totalExpense, accountInfo,addIncome, addExpense and accountBalance inner functions. Incomes is a set of incomes and its description and expenses is also a set of expenses and its description.

 CONGRATULATIONS ! DAY-19 COMPLETED 

DAY-20 Writing Clean Codes

JavaScript Style Guide

A JavaScript style guide is a set of standards that tells how JavaScript code should be written and organized. In this section, we will talk about JavaScript guides and how to write a clean code.

JavaScript is a programming language and like human language it has syntax. The syntax of JavaScript has to be written following a certain style guideline for sake of convince and simplicity.

Why we need style guide

You have been coding alone for so long but now it seems to work in a team. It does not matter in anyway you write you code as long as it running, however when you work in team of 10 or 20 or more developer on one project and on the same code base, the code will be messy and hard to manage if there is no any guidelines to follow.

You can develop your own guidelines and conventions or you can also adapt well developed guidelines. Let us the most common know guidelines. Most common JavaScript Style Guides

- Airbnb JavaScript Style Guide
- JavaScript Standard Style Guide
- Google JavaScript Style Guide

Airbnb JavaScript Style Guide

Airbnb has one of the most popular JavaScript style guides on the internet. It covers nearly every aspect of JavaScript as well and it is adopted by many developer and companies. You may checkout the Airbnb style guide. I would also recommend to try it. Their style is very easy to use and simple to understand.

Standard JavaScript Style Guide

This is guideline is not as popular as Airbnb but it worth to look at it. They removed the semicolon in their style guide.

Google JavaScript Style Guide

I do not say much about Googles guideline and I did not use rather I would suggest you to have a look from this link.

JavaScript Coding Conventions

In this challenge also we used the general JavaScript coding writing conventions and guides. Coding conventions are style guidelines for programming which are developed by an individual, a team or a company.

Coding conventions helps:

- to write clean code
- to improve code readability
- to improve code re-useability and maintainability

Coding conventions includes

- Naming and declaration rules for variables
- Naming and declaration rules for functions
- Rules for the use of white space, indentation, and comments
- Programming practices and principles

Conventions use in 30DaysOfJavaScript

In this challenge we follow the regular JavaScript convention but I added also my preference of writing.

- We used camelCase for variables and functions.
- All variable names start with a letter.

- We chose to use *const* for constants, arrays, objects and functions. In stead of double quote, we chose to use single quote or backtick. Single quote is becoming trendy.
- We also removed semicolons from our code but it is a matter of personal preference.
- Space around arithmetic operators, assignment operators and after comma
- Arrow function instead of function declaration
- Explicit return instead of implicit return if the function is one liner
- No trailing comma in the last value of an object
- We prefer this +=, -=, *= /=, **= instead of the longer version
- When we use console.log() it is good to print with a tag string to identify from where the console is coming

Variables

```
let firstName = 'Asabeneh'
let lastName = 'Yetayeh'
let country = 'Finland'
let city = 'Helsinki'

const PI = Math.PI
const gravity = 9.81
```

Arrays

We chose to make array names plural

- names
- numbers
- countries
- languages
- skills
- fruits
- vegetables

```
// arrays
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']
const numbers = [0, 3.14, 9.81, 37, 98.6, 100]
const countries = ['Finland', 'Denmark', 'Sweden', 'Norway',
'Iceland']
const languages = ['Amharic', 'Arabic', 'English', 'French',
'Spanish']
const skills = ['HTML', 'CSS', 'JavaScript', 'React',
'Python']
const fruits = ['banana', 'orange', 'mango', 'lemon']
const vegetables = ['Tomato', 'Potato', 'Cabbage', 'Onion',
'Carrot']
```

Functions

By now you are very familiar function declaration, expression function, arrow function and anonymous function. In this challenge we tend to use arrow function instead of other functions. Arrow function is not a replacement for other functions. In addition, arrow functions and function declarations are not exactly the same. So you should know when to use and when not. I will cover the difference in detail in other sections. We will use explicit return instead of implicit return if the function is one liner

```
// function which return full name of a person
const printFullName = (firstName, lastName) => firstName + ' '
+ lastName

// function which calculates a square of a number
const square = (n) => n * n

// a function which generate random hexa colors
const hexaColor = () => {
  const str = '0123456789abcdef'
  let hexa = '#'
  let index
  for (let i = 0; i < 6; i++) {
    index = Math.floor(Math.random() * str.length)
    hexa += str[index]
  }
  return hexa
}

// a function which shows date and time
const showDateTime = () => {
  const now = new Date()
  const year = now.getFullYear()
  const month = now.getMonth() + 1
```

```
  const date = now.getDate()
  let hours = now.getHours()
  let minutes = now.getMinutes()
  if (hours < 10) {
    hours = '0' + hours
  }
  if (minutes < 10) {
    minutes = '0' + minutes
  }

  const dateMonthYear = date + '.' + month + '.' + year
  const time = hours + ':' + minutes
  const fullTime = dateMonthYear + ' ' + time
  return fullTime
}
```

The `new Dat().toLocaleString()` can also be used to display current date time. The `toLocaleString()` methods takes different arguments. You may learn more about date and time from this link.

Loops

We coverer many types of loops in this challenges. The regular for loop, while loop, do while loop, for of loop, forEach loop and for in loop. Lets see how we use them:

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

// declaring an array variable
const names = ['Asabeneh', 'Mathias', 'Elias', 'Brook']

// iterating an array using regular for loop
let len = names.length;
for(let i = 0; i < len; i++){
    console.log(names[i].toUpperCase())
}


// iterating an array using for of
for( const name of names) {
    console.log(name.toUpperCase())
}

// iterating array using forEach
names.forEach((name) => name.toUpperCase())
```

```
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills:
['HTML','CSS','JavaScript','React','Node','MongoDB','Python','
D3.js'],
  isMarried: true
}
for(const key in person) {
    console.log(key)
}
```

Objects

We declare object literal with *const*.

```
// declaring object literal
const person = {
  firstName: 'Asabeneh',
  lastName: 'Yetayeh',
  age: 250,
  country: 'Finland',
  city: 'Helsinki',
  skills: ['HTML','CSS','JavaScript','TypeScript',
'React','Node','MongoDB','Python','D3.js'],
  isMarried: true
}
// iterating through object keys
for(const key in person) {
    console.log(key, person[key])
}
```

Conditional

We say if, if else, if else if else, switch and ternary operators in previous challenges.

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

```
// if else
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
```

```
// if else if else if else

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`)
}
```

```
// 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.')
}
```

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

Classes

We declare class with CamelCase which starts with capital letter.

```
// syntax
class ClassName {
    // code goes here
}
// defining class
class Person {
  constructor(firstName, lastName) {
    console.log(this) // Check the output from here
    this.firstName = firstName
    this.lastName = lastName
  }
}
```

Whatever style guide you follow be consistent. Follow some programming paradigms and design patterns. Remember, if you do not write you code in certain order or fashion it will be hard to read your code. So, do a favor for yourself or for someone who is going to read your code by writing readable code.

🍪 You are tidy. Now, you knew how to write clean code, so anyone who know the English language can understand your code. You are always progressing and you are a head of 20 steps to your way to greatness.

🎉 CONGRATULATIONS ! DAY-20 COMPLETED 🎉

DAY-21 DOCUMENT OBJECT MODEL(DOM)-1

Document Object Model (DOM) - Day 1

HTML document is structured as a JavaScript Object. Every HTML element has a different properties which can help to manipulate it. It is possible to get, create, append or remove HTML elements using JavaScript. Check the examples below. Selecting HTML element using JavaScript is similar to selecting using CSS. To select an HTML element, we use tag name, id, class name or other attributes.

Getting Element

We can access already created element or elements using JavaScript. To access or get elements we use different methods. The code below has four *h1* elements. Let us see the different methods to access the *h1* elements.

```
<!DOCTYPE html>
  <html lang="en">
    <head>
      <title>Document Object Model</title>
    </head>
    <body>

     <h1 class='title' id='first-title'>First Title</h1>
     <h1 class='title' id='second-title'>Second Title</h1>
     <h1 class='title' id='third-title'>Third Title</h1>
     <h1></h1>

    </body>
  </html>
```

Getting elements by tag name

getElementsByTagName():takes a tag name as a string parameter and this method returns an HTMLCollection object. An HTMLCollection is an array like object of HTML elements. The length property provides the size of the collection. Whenever we use this method we access the individual elements using index or after loop through each individual items. An HTMLCollection does not support all array methods therefore we should use regular for loop instead of forEach.

```
// syntax
document.getElementsByTagName('tagname')
```

```
const allTitles = document.getElementsByTagName('h1')

console.log(allTitles) //HTMLCollections
console.log(allTitles.length) // 4

for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i]) // prints each elements in the
HTMLCollection
}
```

Getting elements by class name

getElementsByClassName() method returns an HTMLCollection object. An HTMLCollection is an array like list of HTML elements. The length property provides the size of the collection. It is possible to loop through all the HTMLCollection elements. See the example below

```
//syntax
document.getElementsByClassName('classname')
```

```
const allTitles = document.getElementsByClassName('title')

console.log(allTitles) //HTMLCollections
console.log(allTitles.length) // 4

for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i]) // prints each elements in the
HTMLCollection
}
```

Getting an element by id

getElementsById() targets a single HTML element. We pass the id without # as an argument.

```
//syntax
document.getElementById('id')
```

```
let firstTitle = document.getElementById('first-title')
console.log(firstTitle) // <h1>First Title</h1>
```

Getting elements by using querySelector methods

The *document.querySelector* method can select an HTML or HTML elements by tag name, by id or by class name.

querySelector: can be used to select HTML element by its tag name, id or class. If the tag name is used it selects only the first element.

```
let firstTitle = document.querySelector('h1') // select the
first available h1 element
let firstTitle = document.querySelector('#first-title') //
select id with first-title
let firstTitle = document.querySelector('.title') // select
the first available element with class title
```

querySelectorAll: can be used to select html elements by its tag name or class. It returns a nodeList which is an array like object which supports array methods. We can use ***for loop*** or ***forEach*** to loop through each nodeList elements.

```
const allTitles = document.querySelectorAll('h1') # selects
all the available h1 elements in the page

console.log(allTitles.length) // 4
for (let i = 0; i < allTitles.length; i++) {
  console.log(allTitles[i])
}

allTitles.forEach(title => console.log(title))
const allTitles = document.querySelectorAll('.title') // the
same goes for selecting using class
```

Adding attribute

An attribute is added in the opening tag of HTML which gives additional information about the element. Common HTML attributes: id, class, src, style, href,disabled, title, alt. Lets add id and class for the fourth title.

```
const titles = document.querySelectorAll('h1')
titles[3].className = 'title'
titles[3].id = 'fourth-title'
```

Adding attribute using setAttribute

The ***setAttribute()*** method set any html attribute. It takes two parameters the type of the attribute and the name of the attribute. Let's add class and id attribute for the fourth title.

```
const titles = document.querySelectorAll('h1')
titles[3].setAttribute('class', 'title')
titles[3].setAttribute('id', 'fourth-title')
```

Adding attribute without setAttribute

We can use normal object setting method to set an attribute but this can not work for all elements. Some attributes are DOM object property and they can be set directly. For instance id and class

```
//another way to setting an attribute
titles[3].className = 'title'
titles[3].id = 'fourth-title'
```

Adding class using classList

The class list method is a good method to append additional class. It does not override the original class if a class exists rather it adds additional class for the element.

```
//another way to setting an attribute: append the class,
doesn't over ride
titles[3].classList.add('title', 'header-title')
```

Removing class using remove

Similar to adding we can also remove class from an element. We can remove a specific class from an element.

```
//another way to setting an attribute: append the class,
doesn't over ride
titles[3].classList.remove('title', 'header-title')
```

Adding Text to HTML element

An HTML is a build block of an opening tag, a closing tag and a text content. We can add a text content using the property *textContent* or *innerHTML.

Adding Text content using textContent

The *textContent* property is used to add text to an HTML element.

```
const titles = document.querySelectorAll('h1')
titles[3].textContent = 'Fourth Title'
```

Adding Text Content using innerHTML

Most people get confused between *textContent* and *innerHTML*. *textContent* is meant to add text to an HTML element, however innerHTML can add a text or HTML element or elements as a child.

Text Content

We assign *textContent* HTML object property to a text

```
const titles = document.querySelectorAll('h1')
titles[3].textContent = 'Fourth Title'
```

Inner HTML

We use innerHTML property when we like to replace or a completely new children content to a parent element. It value we assign is going to be a string of HTML elements.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript for Everyone:DOM</title>
  </head>
  <body>
```

```
    <div class="wrapper">
        <h1>Asabeneh Yetayeh challenges in 2020</h1>
        <h2>30DaysOfJavaScript Challenge</h2>
        <ul></ul>
    </div>
    <script>
    const lists = `
    <li>30DaysOfPython Challenge Done</li>
            <li>30DaysOfJavaScript Challenge Ongoing</li>
            <li>30DaysOfReact Challenge Coming</li>
            <li>30DaysOfFullStack Challenge Coming</li>
            <li>30DaysOfDataAnalysis Challenge Coming</li>
            <li>30DaysOfReactNative Challenge Coming</li>
            <li>30DaysOfMachineLearning Challenge Coming</li>`
  const ul = document.querySelector('ul')
  ul.innerHTML = lists
    </script>
  </body>
</html>
```

The innerHTML property can allow us also to remove all the children of a parent element. Instead of using removeChild() I would recommend the following method.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript for Everyone:DOM</title>
  </head>
  <body>
    <div class="wrapper">
        <h1>Asabeneh Yetayeh challenges in 2020</h1>
        <h2>30DaysOfJavaScript Challenge</h2>
        <ul>
            <li>30DaysOfPython Challenge Done</li>
            <li>30DaysOfJavaScript Challenge Ongoing</li>
            <li>30DaysOfReact Challenge Coming</li>
            <li>30DaysOfFullStack Challenge Coming</li>
            <li>30DaysOfDataAnalysis Challenge Coming</li>
            <li>30DaysOfReactNative Challenge Coming</li>
            <li>30DaysOfMachineLearning Challenge Coming</li>
        </ul>
    </div>
    <script>
  const ul = document.querySelector('ul')
  ul.innerHTML = ''
    </script>
  </body>
</html>
```

Adding style

Adding Style Color

Let us add some style to our titles. If the element has even index we give it green color else red.

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px
font size
  if (i % 2 === 0) {
    title.style.color = 'green'
  } else {
    title.style.color = 'red'
  }
})
```

Adding Style Background Color

Let us add some style to our titles. If the element has even index we give it green color else red.

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px
font size
  if (i % 2 === 0) {
    title.style.backgroundColor = 'green'
  } else {
    title.style.backgroundColor = 'red'
  }
})
```

Adding Style Font Size

Let us add some style to our titles. If the element has even index we give it 20px else 30px

```
const titles = document.querySelectorAll('h1')
titles.forEach((title, i) => {
  title.style.fontSize = '24px' // all titles will have 24px
font size
  if (i % 2 === 0) {
    title.style.fontSize = '20px'
  } else {
    title.style.fontSize = '30px'
  }
})
```

As you have notice, the properties of css when we use it in JavaScript is going to be a camelCase. The following CSS properties change from background-color to backgroundColor, font-size to fontSize, font-family to fontFamily, margin-bottom to marginBottom.

🎉 Now, you are fully charged with a super power, you have completed the most important and challenging part of the challenge and in general JavaScript. You learned DOM and now you have the capability to build and develop applications. Now do some exercises for your brain and for your muscle.

Exercises

Exercise: Level 1

1. Create an index.html file and put four p elements as above: Get the first paragraph by using ***document.querySelector(tagname)*** and tag name
2. Get each of the the paragraph using ***document.querySelector('#id')*** and by their id
3. Get all the p as nodeList using ***document.querySelectorAll(tagname)*** and by their tag name
4. Loop through the nodeList and get the text content of each paragraph
5. Set a text content to paragraph the fourth paragraph,***Fourth Paragraph***
6. Set id and class attribute for all the paragraphs using different attribute setting methods

Exercise: Level 2

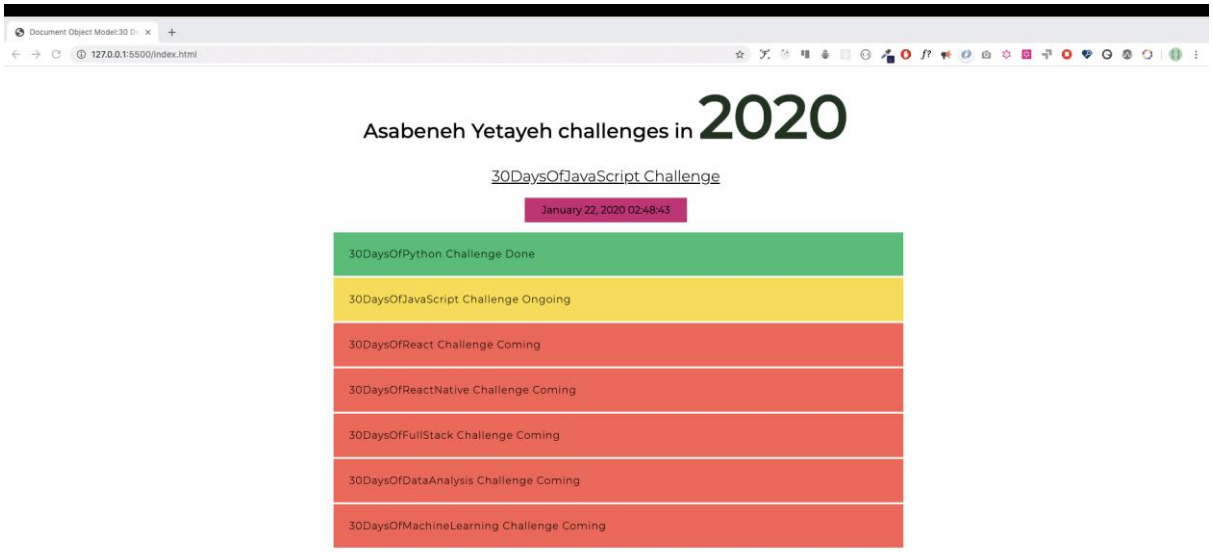
1. Change stye of each paragraph using JavaScript(eg. color, background, border, font-size, font-family)
2. Select all paragraphs and loop through each elements and give the first and third paragraph a color of green, and the second and the fourth paragraph a red color
3. Set text content, id and class to each paragraph

Exercise: Level 3

DOM: Mini project 1

1. Develop the following application, use the following HTML elements to get started with. You will get the same code on starter folder. Apply all the styles and functionality using JavaScript only.
    - The year color is changing every 1 second
    - The date and time background color is changing every on seconds
    - Completed challenge has background green
    - Ongoing challenge has background yellow
    - Coming challenges have background red

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript for Everyone:DOM</title>
  </head>
  <body>
    <div class="wrapper">
        <h1>Asabeneh Yetayeh challenges in 2020</h1>
        <h2>30DaysOfJavaScript Challenge</h2>
        <ul>
            <li>30DaysOfPython Challenge Done</li>
            <li>30DaysOfJavaScript Challenge Ongoing</li>
            <li>30DaysOfReact Challenge Coming</li>
            <li>30DaysOfFullStack Challenge Coming</li>
            <li>30DaysOfDataAnalysis Challenge Coming</li>
            <li>30DaysOfReactNative Challenge Coming</li>
            <li>30DaysOfMachineLearning Challenge Coming</li>
        </ul>
    </div>
  </body>
</html>
```



🎉 CONGRATULATIONS ! DAY-21 COMPLETED 🎉

DAY-22 DOCUMENT OBJECT MODEL(DOM)-2

Creating an Element

To create an HTML element we use tag name. Creating an HTML element using JavaScript is very simple and straight forward. We use the method *document.createElement()*. The method takes an HTML element tag name as a string parameter.

```
// syntax
document.createElement('tagname')
```

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>

    <script>
        let title = document.createElement('h1')
        title.className = 'title'
        title.style.fontSize = '24px'
        title.textContent = 'Creating HTML element DOM Day 2'

        console.log(title)
    </script>
</body>

</html>
```

Creating elements

To create multiple elements we should use loop. Using loop we can create as many HTML elements as we want. After we create the element we can assign value to the different properties of the HTML object.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>

    <script>
        let title
        for (let i = 0; i < 3; i++) {
            title = document.createElement('h1')
            title.className = 'title'
            title.style.fontSize = '24px'
            title.textContent = i
            console.log(title)
        }
    </script>
</body>

</html>
```

Appending child to a parent element

To see a created element on the HTML document we should append it to the parent as a child element. We can access the HTML document body using *document.body*. The *document.body* support the *appendChild()* method. See the example below.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>

    <script>
        // creating multiple elements and appending to parent
element
        let title
```

```
        for (let i = 0; i < 3; i++) {
            title = document.createElement('h1')
            title.className = 'title'
            title.style.fontSize = '24px'
            title.textContent = i
            document.body.appendChild(title)
        }
    </script>
</body>
</html>
```

Removing a child element from a parent node

After creating an HTML, we may want to remove element or elements and we can use the *removeChild()* method.

Example:

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Removing child Node</h1>
    <h2>Asabeneh Yetayeh challenges in 2020</h1>
    <ul>
        <li>30DaysOfPython Challenge Done</li>
        <li>30DaysOfJavaScript Challenge Done</li>
        <li>30DaysOfReact Challenge Coming</li>
        <li>30DaysOfFullStack Challenge Coming</li>
        <li>30DaysOfDataAnalysis Challenge Coming</li>
        <li>30DaysOfReactNative Challenge Coming</li>
        <li>30DaysOfMachineLearning Challenge Coming</li>
    </ul>

    <script>
        const ul = document.querySelector('ul')
        const lists = document.querySelectorAll('li')
        for (const list of lists) {
            ul.removeChild(list)

        }
    </script>
</body>
</html>
```

As we have see in the previous section there is a better way to eliminate all the inner HTML elements or the children of a parent element using the method *innerHTML* properties.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Removing child Node</h1>
    <h2>Asabeneh Yetayeh challenges in 2020</h1>
    <ul>
        <li>30DaysOfPython Challenge Done</li>
        <li>30DaysOfJavaScript Challenge Done</li>
        <li>30DaysOfReact Challenge Coming</li>
        <li>30DaysOfFullStack Challenge Coming</li>
        <li>30DaysOfDataAnalysis Challenge Coming</li>
        <li>30DaysOfReactNative Challenge Coming</li>
        <li>30DaysOfMachineLearning Challenge Coming</li>
    </ul>

    <script>
        const ul = document.querySelector('ul')
        ul.innerHTML = ''
    </script>
</body>

</html>
```

The above snippet of code cleared all the child elements.

You are so special, you are progressing everyday. Now, you knew how to destroy a created DOM element when it is needed. You learned DOM and now you have the capability to build and develop applications. You are left with only eight days to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

Exercises: Level 1

1. Create a div container on HTML document and create 100 to 100 numbers dynamically and append to the container div.
    - Even numbers background is green
    - Odd numbers background is yellow
    - Prime numbers background is red

    (DEMO IMAGE ON THE NEXT PAGE)

Number Generator

30DaysOfJavaScript:DOM Day 2

Author: Asabeneh Yetayeh

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59
60	61	62	63	64	65
66	67	68	69	70	71
72	73	74	75	76	77
78	79	80	81	82	83
84	85	86	87	88	89
90	91	92	93	94	95
96	97	98	99	100	101

CONGRATULATIONS ! DAY-22 COMPLETED

DAY-23 EVENT LISTENERS

Event Listeners

Common HTML events:onclick, onchange, onmouseover, onmouseout, onkeydown, onkeyup, onload. We can add event listener method to any DOM object. We use ***addEventListener()*** method to listen different event types on HTML elements. The *addEventListener()* method takes two arguments, an event listener and a callback function.

```
selectedElement.addEventListener('eventlistner', function(e) {
  // the activity you want to occur after the event will be in
here
})
// or

selectedElement.addEventListener('eventlistner', e => {
  // the activity you want to occur after the event will be in
here
})
```

Click

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event.

Example: click

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button>Click Me</button>

    <script>
      const button = document.querySelector('button')
      button.addEventListener('click', e => {
        console.log('e gives the event listener object:', e)
```

```
        console.log('e.target gives the selected element: ',
e.target)
        console.log(
          'e.target.textContent gives content of selected
element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
</html>
```

An event can be also attached directly to the HTML element as inline script.

Example: onclick

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button onclick="clickMe()">Click Me</button>
    <script>
      const clickMe = () => {
        alert('We can attach event on HTML element')
      }
    </script>
  </body>
</html>
```

Double Click

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event. **Example: dblclick**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
```

```
    <button>Click Me</button>
    <script>
      const button = document.querySelector('button')
      button.addEventListener('dblclick', e => {
        console.log('e gives the event listener object:', e)
        console.log('e.target gives the selected element: ',
e.target)
        console.log(
          'e.target.textContent gives content of selected
element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
</html>
```

Mouse enter

To attach an event listener to an element, first we select the element then we attach the addEventListener method. The event listener takes event type and callback functions as argument.

The following is an example of click type event.

Example: mouseenter

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
  </head>

  <body>
    <button>Click Me</button>
    <script>
      const button = document.querySelector('button')
      button.addEventListener('mouseenter', e => {
        console.log('e gives the event listener object:', e)
        console.log('e.target gives the selected element: ',
e.target)
        console.log(
          'e.target.textContent gives content of selected
element: ',
          e.target.textContent
        )
      })
    </script>
  </body>
```

```
</html>
```

By now you are familiar with addEventListen method and how to attach event listener. There are many types of event listeners. But in this challenge we will focus the most common important events. List of events:

- click - when the element clicked
- dblclick - when the element double clicked
- mouseenter - when the mouse point enter to the element
- mouseleave - when the mouse pointer leave the element
- mousemove - when the mouse pointer move on the element
- mouseover - when the mouse pointer move on the element
- mouseout -when the mouse pointer out from the element
- input -when value enter to input field
- change -when value change on input field
- blur -when the element is not focused
- keydown - when a key is down
- keyup - when a key is up
- keypress - when we press any key
- onload - when the browser has finished loading a page

Test the above event types by replacing event type in the above snippet code.

Getting value from an input element

We usually fill forms and forms accept data. Form fields are created using input HTML element. Let us build a small application which allow us to calculate body mas index of a person using two input fields, one button and one p tag.

input value

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

  <body>
    <h1>Body Mass Index Calculator</h1>

    <input type="text" id="mass" placeholder="Mass in
Kilogram" />
    <input type="text" id="height" placeholder="Height in
meters" />
    <button>Calculate BMI</button>

    <script>
      const mass = document.querySelector('#mass')
      const height = document.querySelector('#height')
      const button = document.querySelector('button')

      let bmi
      button.addEventListener('click', () => {
        bmi = mass.value / height.value ** 2
        alert(`your bmi is ${bmi.toFixed(2)}`)
        console.log(bmi)
      })
    </script>
  </body>
</html>
```

input event and change

In the above example, we managed to get input values from two input fields by clicking button. How about if we want to get value without click the button. We can use the *change* or *input* event type to get data right away from the input field when the field is on focus. Let us see how we will handle that.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

  <body>
    <h1>Data Binding using input or change event</h1>

    <input type="text" placeholder="say something" />
    <p></p>

    <script>
      const input = document.querySelector('input')
      const p = document.querySelector('p')

      input.addEventListener('input', e => {
        p.textContent = e.target.value
      })
    </script>
  </body>
</html>
```

blur event

In contrast to *input* or *change*, the *blur* event occur when the input field is not on focus.

```
<!DOCTYPE html>
<html>

<head>
    <title>Document Object Model:30 Days Of JavaScript</title>
</head>

<body>
    <h1>Giving feedback using blur event</h1>

    <input type="text" id="mass" placeholder="say something"
/>
    <p></p>
```

```
    <script>
        const input = document.querySelector('input')
        const p = document.querySelector('p')

        input.addEventListener('blur', (e) => {
            p.textContent = 'Field is required'
            p.style.color = 'red'

        })
    </script>
</body>

</html>
```

keypress, keydow and keyup

We can access all the key numbers of the keyboard using different event listener types. Let us use keypress and get the keyCode of each keyboard keys.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model:30 Days Of JavaScript</title>
  </head>

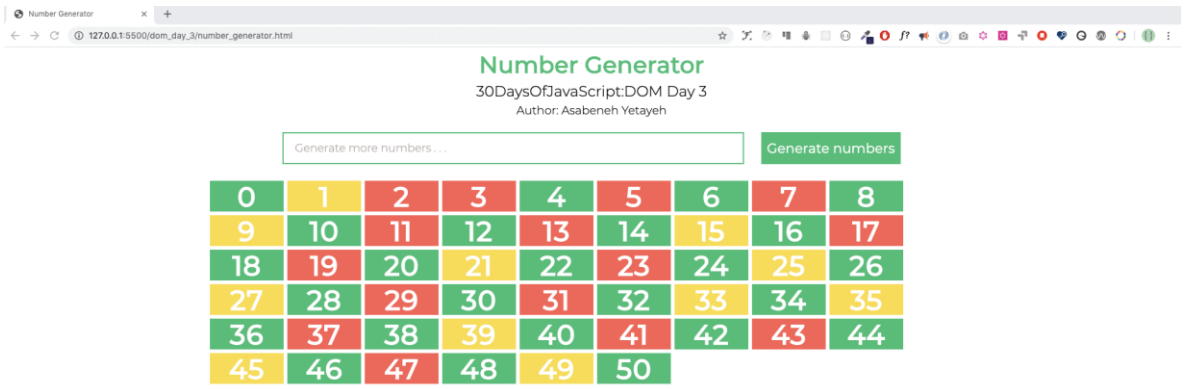
  <body>
    <h1>Key events: Press any key</h1>

    <script>
      document.body.addEventListener('keypress', e => {
        alert(e.keyCode)
      })
    </script>
  </body>
</html>
```

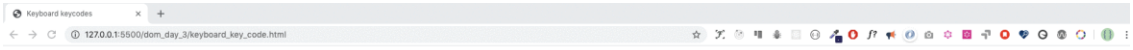
🛞 You are so special, you are progressing everyday. Now, you knew how handle any kind of DOM events. . You are left with only seven days to your way to greatness. Now do some exercises for your brain and for your muscle.

Exercises

1. Generating numbers and marking evens, odds and prime numbers with three different colors. See the image below.



1. Generating the keyboard code code using even listener. The image below.



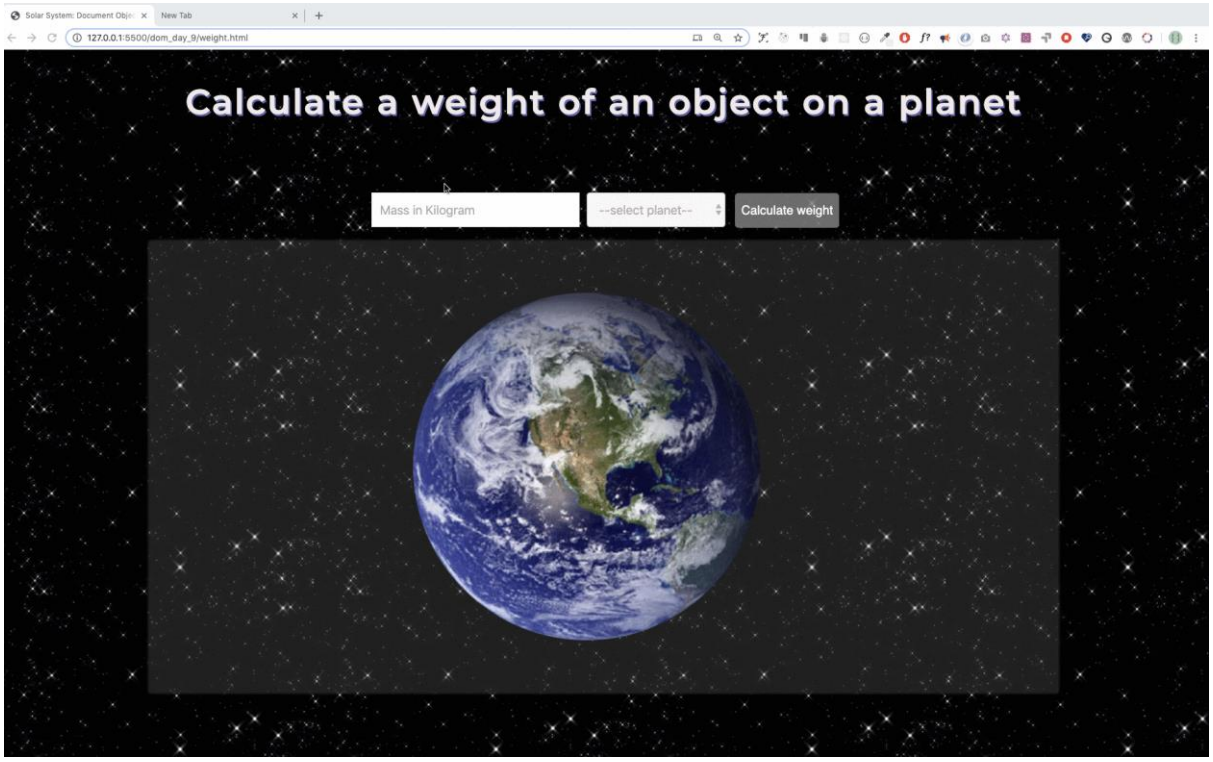
🎉 CONGRATULATIONS ! DAY 23 COMPLETED 🎉

DAY-24 Mini Project Solar System

Exercises

Exercise: Level 1

1. Develop a small application which calculate a weight of an object in a certain planet. The gif image is not complete check the video in the starter file.



CONGRATULATIONS ! DAY-24 COMPLETED

DAY-25 World Countries Data Visualization

Exercise:

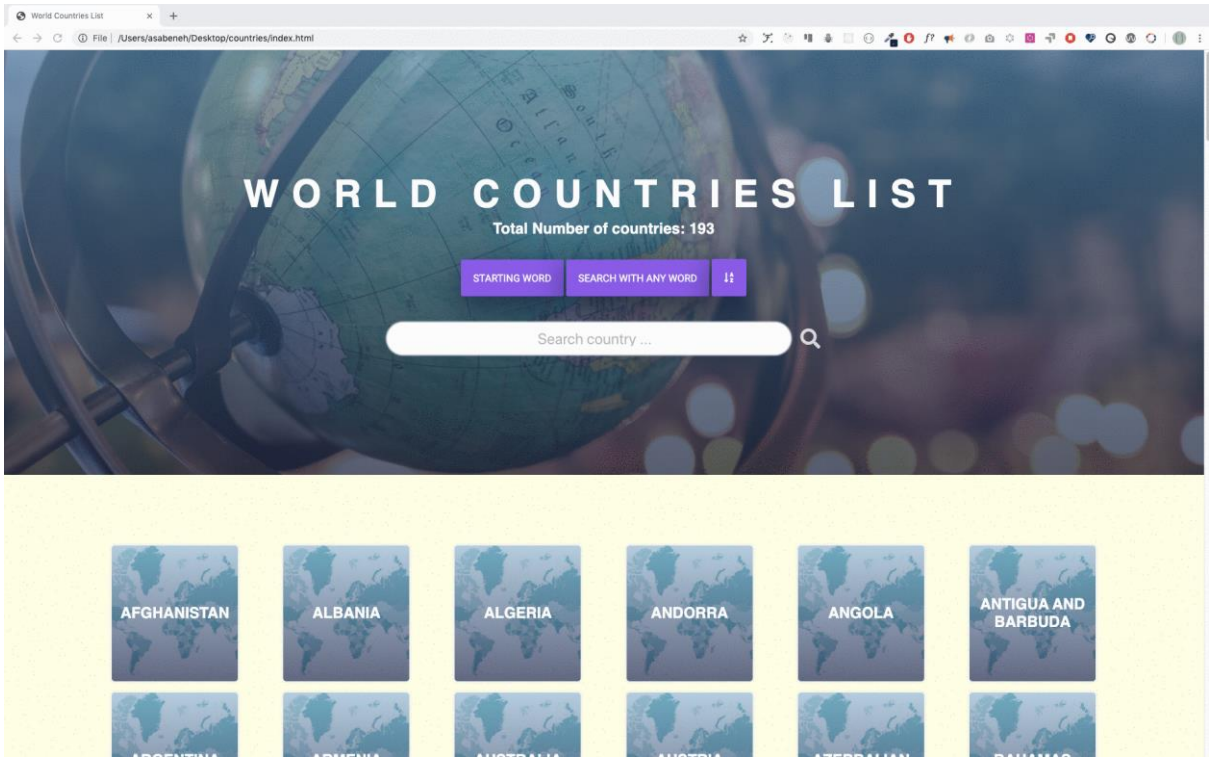
1. Visualize the ten most populated countries and the ten most spoken languages in the world using DOM(HTML, CSS, JS)

🎉 CONGRATULATIONS ! DAY-25 COMPLETED 🎉

DAY-26 World Countries Data Visualization 2

Exercise

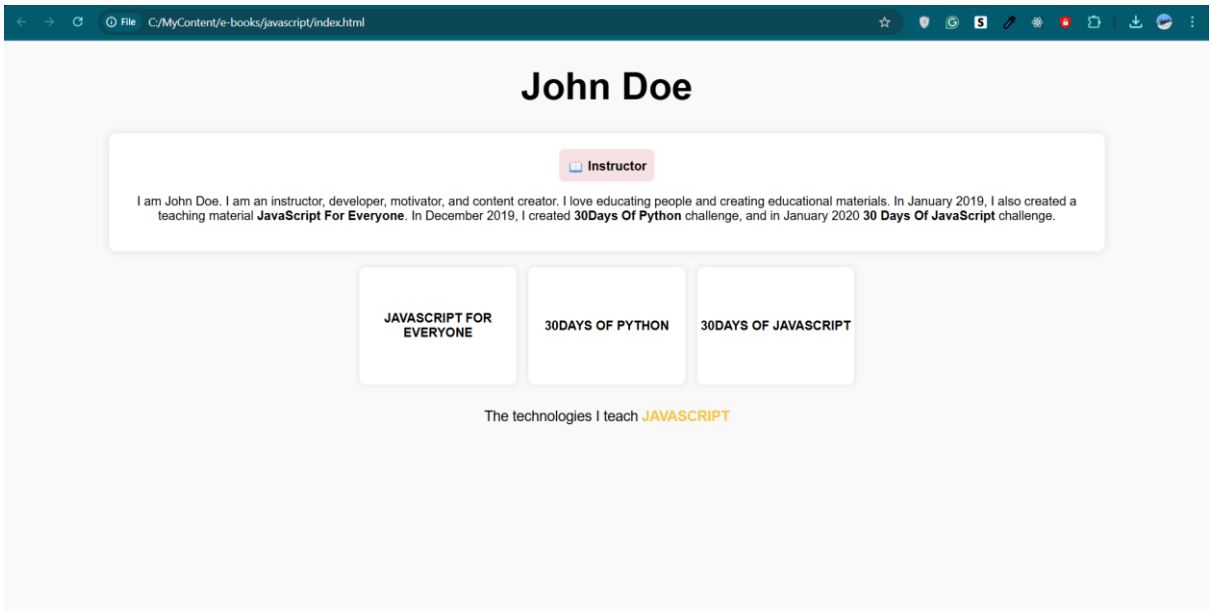
1. Visualize the countries array as follows



🎉 CONGRATULATIONS ! DAY-26 COMPLETED 🎉

DAY-27 PORTFOLIO

1. Create the following using HTML, CSS, and JavaScript

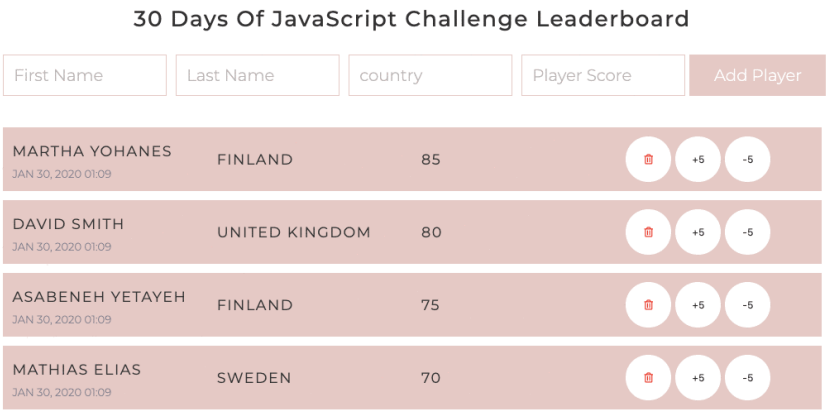


🎉 CONGRATULATIONS ! DAY-27 🎉

DAY-28 PROJECT LEADERBOARD

Exercise

1. Create the following using HTML, CSS, and JavaScript

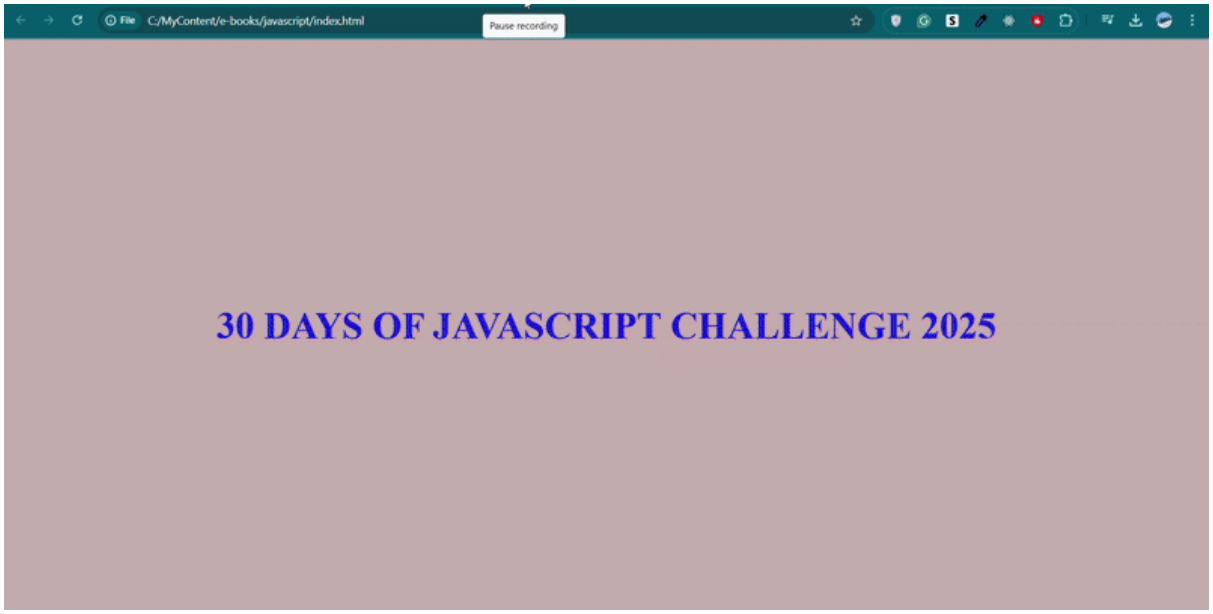


🎉 CONGRATULATIONS ! DAY-28 COMPLETED 🎉

DAY-29 ANIMATING CHARACTERS

Exercise

1. Create the following using HTML, CSS, and JavaScript



🎉 CONGRATULATIONS ! DAY-29 COMPLETED 🎉

DAY-30 FINAL PROJECT

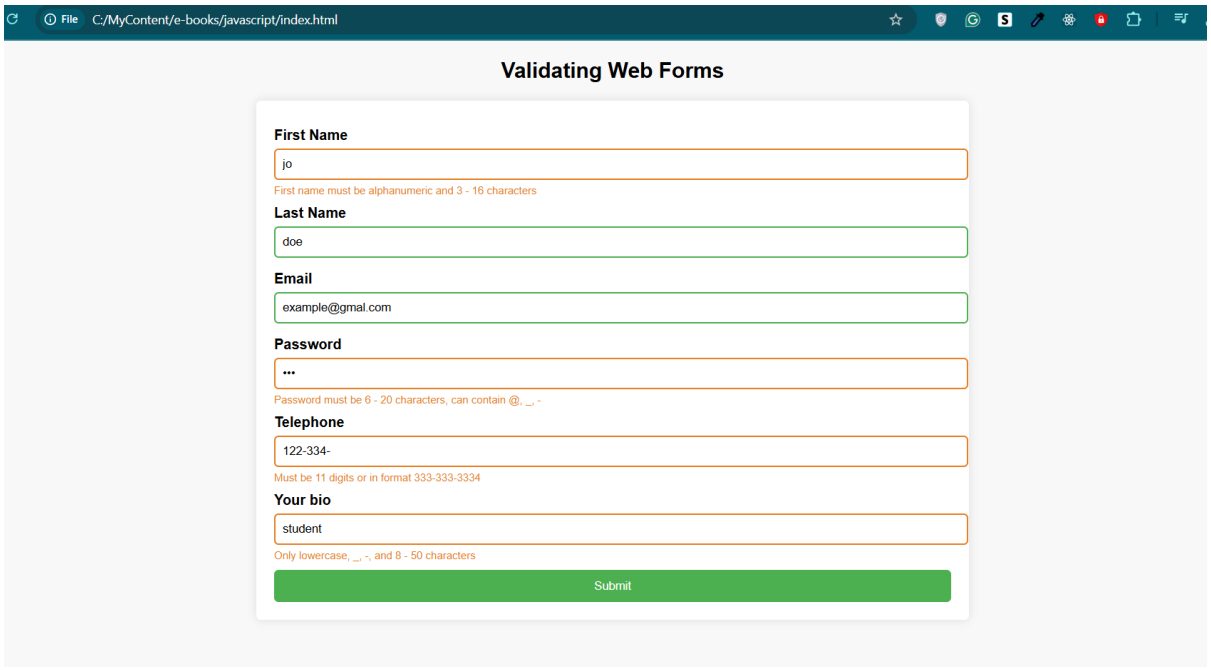
Exercises

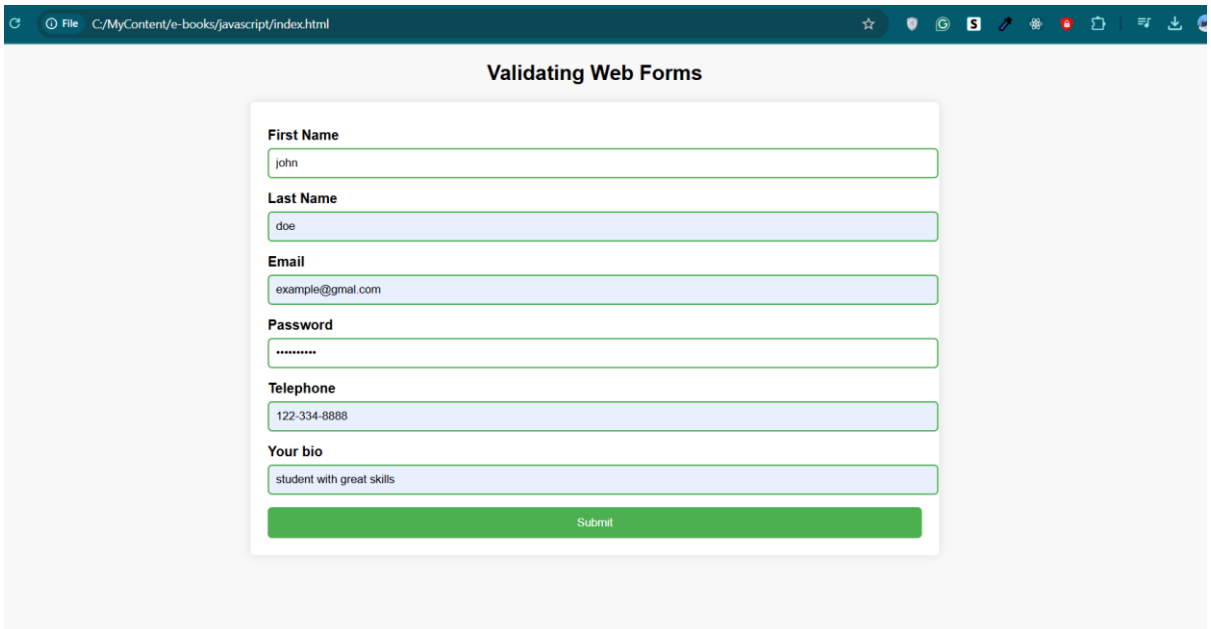
Exercise: Level 1

1. Create the following using HTML, CSS, and JavaScript

Exercise: Level 2

Validate the following form using regex.





🎉 CONGRATULATIONS ! DAY-30 COMPLETED 🎉

100+ JavaScript Projects to Practice

Congratulations on completing the **30 Days of JavaScript** journey! 🎉 You've built a solid foundation in JavaScript, covering core concepts, best practices, and hands-on exercises. Now, it's time to take your skills to the next level by working on real-world projects.

Why Build Projects? 💡

Practicing with projects is the most effective way to reinforce what you've learned. Here's how it helps: ✅ **Solidify your understanding** of JavaScript concepts by applying them in real scenarios.

✅ **Enhance your problem-solving skills** by tackling practical coding challenges.

✅ **Build a strong portfolio** to showcase your skills to potential employers.

✅ **Gain confidence** in working with real-world applications.

What's Inside This Repository? 🗄️

This carefully curated collection includes **100+ JavaScript projects** ranging from beginner-friendly exercises to advanced real-world applications. You'll explore concepts like:

🖼️ DOM Manipulation

🌟 ES6+ Features

📎 API Integration

▯ Data Structures & Algorithms

🎯 Event Handling

⏳ Asynchronous JavaScript (Promises, Async/Await)

💾 Browser Storage (LocalStorage, SessionStorage)

Get Started Now! 🚀

Start coding today by exploring the full list of projects here:

📎 **100+ JavaScript Projects GitHub Repository**

Each project comes with step-by-step instructions to help you build and improve. Happy coding! 💻✨

🛠 How to Use

1. **Clone this repository:**

```
git clone https://github.com/pradipchaudhary/100-js-projects.git
```

2. **Navigate to the project folder:**

```
cd 100-js-projects
```

3. **Each project is stored in its own directory with its own README file for instructions. Simply open the index.html file in a browser to view the project or open the project folder in your code editor to start coding.**

🌟 Congratulations on reaching this milestone! Your dedication and effort have brought you to a whole new level of mastery. You've proven what it takes to get here, and your achievement speaks for itself. You are truly remarkable!

Now, take a moment to celebrate your success—whether with friends, family, or simply by acknowledging how far you've come. This is just the beginning of your journey, and I look forward to seeing you take on new challenges in the future! 🚀