**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

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.')

### 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 neds 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.**

1. **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.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.**

1. 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.
2. let a = 4

let b = 3

4 is greater than 3

1. Even numbers are divisible by 2 and the remainder is zero. How do you check, if a number is even or not using JavaScript?
2. Enter a number: 2
3. 2 is an even number
4. 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.**

## 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(' ')

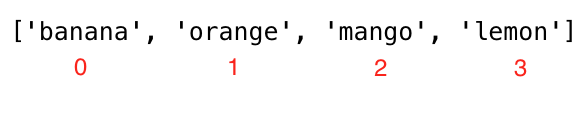
console.log(words)

// 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.

[](https://github.com/Asabeneh/30-Days-Of-JavaScript/blob/master/images/array_index.png)

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 exist in the array')**

**} else {**

**console.log('This fruit does not exist in the array')**

**}**

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

**// we can use also ternary here**

**index != -1 ? console.log('This fruit does exist in the array'): console.log('This fruit does not exist in the array'**)

**// let us check if a 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 exist in the array')**

**} else {**

**console.log('This fruit does not 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]**

**console.log(numbers.splice()) // -> remove all items**

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

**console.log(numbers.splice(0,1)) // remove the first item**

**const numbers = [1, 2, 3, 4, 5, 6];**

**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"]

### 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
3. let text =
4. 'I love teaching and empowering people. I teach HTML, CSS, JS, React, Python.'
5. console.log(words)

console.log(words.length)

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

13

1. 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'

1. 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.
2. 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.
3. Concatenate the following two variables and store it in a fullStack variable.
4. const frontEnd = ['HTML', 'CSS', 'JS', 'React', 'Redux']
5. 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](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js)

**Loops**

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

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

**for Loop**

// For loop structure

for(initialization, condition, increment/decrement){

// code goes here

}

for(let i = 0; i <= 5; i++){

console.log(i)

}

// 0 1 2 3 4 5

for(let i = 5; i >= 0; i--){

console.log(i)

}

// 5 4 3 2 1 0

for(let i = 0; i <= 5; i++){

console.log(`${i} \* ${i} = ${i \* i}`)

}

0 \* 0 = 0

1 \* 1 = 1

2 \* 2 = 4

3 \* 3 = 9

4 \* 4 = 16

5 \* 5 = 25

const countries = ['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

}

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. #
6. ##
7. ###
8. ####
9. #####
10. ######

#######

1. Use loop to print the following pattern:
2. 0 x 0 = 0
3. 1 x 1 = 1
4. 2 x 2 = 4
5. 3 x 3 = 9
6. 4 x 4 = 16
7. 5 x 5 = 25
8. 6 x 6 = 36
9. 7 x 7 = 49
10. 8 x 8 = 64
11. 9 x 9 = 81

10 x 10 = 100

1. Using loop print the following pattern
2. i i^2 i^3
3. 0 0 0
4. 1 1 1
5. 2 4 8
6. 3 9 27
7. 4 16 64
8. 5 25 125
9. 6 36 216
10. 7 49 343
11. 8 64 512
12. 9 81 729

10 100 1000

1. Use for loop to iterate from 0 to 100 and print only even numbers
2. Use for loop to iterate from 0 to 100 and print only odd numbers
3. Use for loop to iterate from 0 to 100 and print only prime numbers
4. 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.

1. 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.

1. 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]

1. Develop a small script which generate array of 5 random numbers
2. Develop a small script which generate array of 5 random numbers and the numbers must be unique
3. 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

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

'#ee33df'

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

rgb(240,180,80)

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

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

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

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

1. Use the countries array to create the following array of arrays:
2. [
3. ['Albania', 'ALB', 7],
4. ['Bolivia', 'BOL', 7],
5. ['Canada', 'CAN', 6],
6. ['Denmark', 'DEN', 7],
7. ['Ethiopia', 'ETH', 8],
8. ['Finland', 'FIN', 7],
9. ['Germany', 'GER', 7],
10. ['Hungary', 'HUN', 7],
11. ['Ireland', 'IRE', 7],
12. ['Iceland', 'ICE', 7],
13. ['Japan', 'JAP', 5],
14. ['Kenya', 'KEN', 5]

]

1. 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']

1. 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']

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

Ethiopia

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

['Japan', 'Kenya']

1. Find the longest word in the webTechs array
2. 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]]

1. An application created using MongoDB, Express, React and Node is called a MERN stack app. Create the acronym MERN by using the array mernStack
2. 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.
3. This is a fruit array , ['banana', 'orange', 'mango', 'lemon'] reverse the order using loop without using a reverse method.
4. Print all the elements of array as shown below.
5. const fullStack = [
6. ['HTML', 'CSS', 'JS', 'React'],
7. ['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](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js) and print it as array
5. Find the country containing the hightest number of characters in the [countries array](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js)
6. Extract all the countries contain the word 'land' from the [countries array](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js) and print it as array
7. Extract all the countries containing only four characters from the [countries array](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js) and print it as array
8. Extract all the countries containing two or more words from the [countries array](https://github.com/Asabeneh/30DaysOfJavaScript/tree/master/data/countries.js) and print it as array

**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 an 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(id = null) {

if(id){

}else{

}

//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

**💻 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 *convertCelciusToFahrenheit*.
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.
16. console.log(findMax(0, 10, 5))
17. 10
18. 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*.
3. console.log(solveQuadratic()) // {0}
4. console.log(solveQuadratic(1, 4, 4)) // {-2}
5. console.log(solveQuadratic(1, -1, -2)) // {2, -1}
6. console.log(solveQuadratic(1, 7, 12)) // {-3, -4}
7. console.log(solveQuadratic(1, 0, -4)) //{2, -2}

console.log(solveQuadratic(1, -1, 0)) //{1, 0}

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

08/01/2020 04:08

1. Declare a function name *swapValues*. This function swaps value of x to y.
2. swapValues(3, 4) // x => 4, y=>3

swapValues(4, 5) // x = 5, y = 4

1. Declare a function name *reverseArray*. It takes array as a parameter and it returns the reverse of the array (don't use method).
2. console.log(reverseArray([1, 2, 3, 4, 5]))
3. //[5, 4, 3, 2, 1]
4. console.log(reverseArray(['A', 'B', 'C']))

//['C', 'B', 'A']

1. Declare a function name *capitalizeArray*. It takes array as a parameter and it returns the - capitalizedarray.
2. Declare a function name *addItem*. It takes an item parameter and it returns an array after adding the item
3. Declare a function name *removeItem*. It takes an index parameter and it returns an array after removing an item
4. Declare a function name *sumOfNumbers*. It takes a number parameter and it adds all the numbers in that range.
5. Declare a function name *sumOfOdds*. It takes a number parameter and it adds all the odd numbers in that - range.
6. Declare a function name *sumOfEven*. It takes a number parameter and it adds all the even numbers in that - range.
7. Declare a function name evensAndOdds . It takes a positive integer as parameter and it counts number of evens and odds in the number.
8. evensAndOdds(100);
9. The number of odds are 50.

The number of evens are 51.

1. Write a function which takes any number of arguments and return the sum of the arguments
2. sum(1, 2, 3) // -> 6

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

1. Writ a function which generates a *randomUserIp*.
2. Write a function which generates a *randomMacAddress*
3. Declare a function name *randomHexaNumberGenerator*. When this function is called it generates a random hexadecimal number. The function return the hexadecimal number.
4. console.log(randomHexaNumberGenerator());

'#ee33df'

1. Declare a function name *userIdGenerator*. When this function is called it generates seven character id. The function return the id.
2. 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.
2. userIdGeneratedByUser()
3. 'kcsy2
4. SMFYb
5. bWmeq
6. ZXOYh
7. 2Rgxf
8. '
9. userIdGeneratedByUser()
10. '1GCSgPLMaBAVQZ26
11. YD7eFwNQKNs7qXaT
12. ycArC5yrRupyG00S
13. UbGxOFI7UXSWAyKN
14. dIV0SSUTgAdKwStr

'

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

rgb(125,244,255)

1. Write a function ***arrayOfHexaColors*** which return any number of hexadecimal colors in an array.
2. Write a function ***arrayOfRgbColors*** which return any number of RGB colors in an array.
3. Write a function ***convertHexaToRgb*** which converts hexa color to rgb and it returns an rgb color.
4. Write a function ***convertRgbToHexa*** which converts rgb to hexa color and it returns an hexa color.
5. Write a function ***generateColors*** which can generate any number of hexa or rgb colors.
6. console.log(generateColors('hexa', 3)) // ['#a3e12f', '#03ed55', '#eb3d2b']
7. console.log(generateColors('hexa', 1)) // '#b334ef'
8. 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)'

1. Call your function *shuffleArray*, it takes an array as a parameter and it returns a shuffled array
2. Call your function *factorial*, it takes a whole number as a parameter and it return a factorial of the number
3. Call your function *isEmpty*, it takes a parameter and it checks if it is empty or not
4. Call your function *sum*, it takes any number of arguments and it returns the sum.
5. 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.
6. 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.
7. 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'

1. Write a function called *isPrime*, which checks if a number is prime number.
2. Write a functions which checks if all items are unique in the array.
3. Write a function which checks if all the items of the array are the same data type.
4. 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.
5. Write a function which returns array of seven random numbers in a range of 0-9. All the numbers must be unique.
6. sevenRandomNumbers()

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

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

**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)

// 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'])

// 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) //['name', 'age', 'country', 'skills', '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'))

**💻 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.
3. const users = {
4. Alex: {
5. email: 'alex@alex.com',
6. skills: ['HTML', 'CSS', 'JavaScript'],
7. age: 20,
8. isLoggedIn: false,
9. points: 30
10. },
11. Asab: {
12. email: 'asab@asab.com',
13. skills: ['HTML', 'CSS', 'JavaScript', 'Redux', 'MongoDB', 'Express', 'React', 'Node'],
14. age: 25,
15. isLoggedIn: false,
16. points: 50
17. },
18. Brook: {
19. email: 'daniel@daniel.com',
20. skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux'],
21. age: 30,
22. isLoggedIn: true,
23. points: 50
24. },
25. Daniel: {
26. email: 'daniel@alex.com',
27. skills: ['HTML', 'CSS', 'JavaScript', 'Python'],
28. age: 20,
29. isLoggedIn: false,
30. points: 40
31. },
32. John: {
33. email: 'john@john.com',
34. skills: ['HTML', 'CSS', 'JavaScript', 'React', 'Redux', 'Node.js'],
35. age: 20,
36. isLoggedIn: true,
37. points: 50
38. },
39. Thomas: {
40. email: 'thomas@thomas.com',
41. skills: ['HTML', 'CSS', 'JavaScript', 'React'],
42. age: 20,
43. isLoggedIn: false,
44. points: 40
45. },
46. Paul: {
47. email: 'paul@paul.com',
48. skills: ['HTML', 'CSS', 'JavaScript', 'MongoDB', 'Express', 'React', 'Node'],
49. age: 20,
50. isLoggedIn: false,
51. points: 40
52. }

}```

1. Find people who are MERN stack developer from the users object
2. Set your name in the users object without modifying the original users object
3. Get all keys or properties of users object
4. Get all the values of users object
5. 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

1. 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
2. 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.

## 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>

<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 take 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) //HTMCollections

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) //HTMCollections

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 h2 element

let firstTitle = document.querySelector('#first-title') // select id with first-title

let firstTitle = document.querySelector('.title') // select the first available h2 element with class title

**querySelectorAll**: can be used to select html element by its tag name or class. It return a nodeList which is an array like object which support array methods. We can use **for loop** or **forEach** to loop through each nodeList elements.

const allTitles = document.querySelectorAll('h1')

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].class = '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>

<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>

<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.

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 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>

<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>

## DOM(Document Object Model)-Day 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.

## 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