

# REFUERZO JAVASCRIPT

JONATHAN LÓPEZ LONDOÑO

JLOPEZL@UAO.EDU.CO

315 926 5443



<https://developer.mozilla.org/es/docs/Web/JavaScript>



<https://nodejs.org/es/>



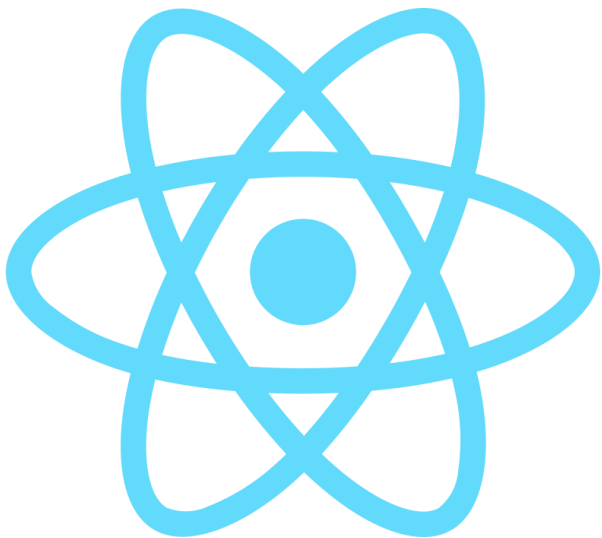
<https://developer.mozilla.org/es/docs/Web/HTML>

<https://developer.mozilla.org/es/docs/Web/CSS>



<https://code.visualstudio.com/>

# REQUIREMENTS TO START



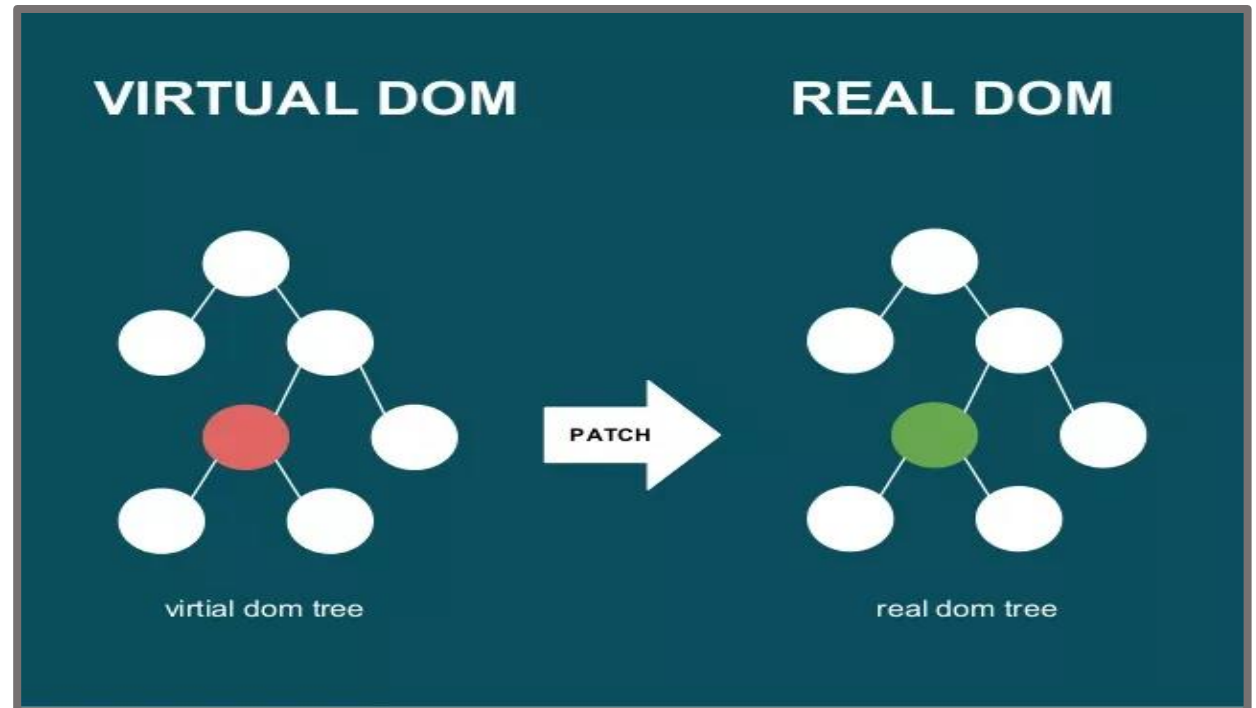
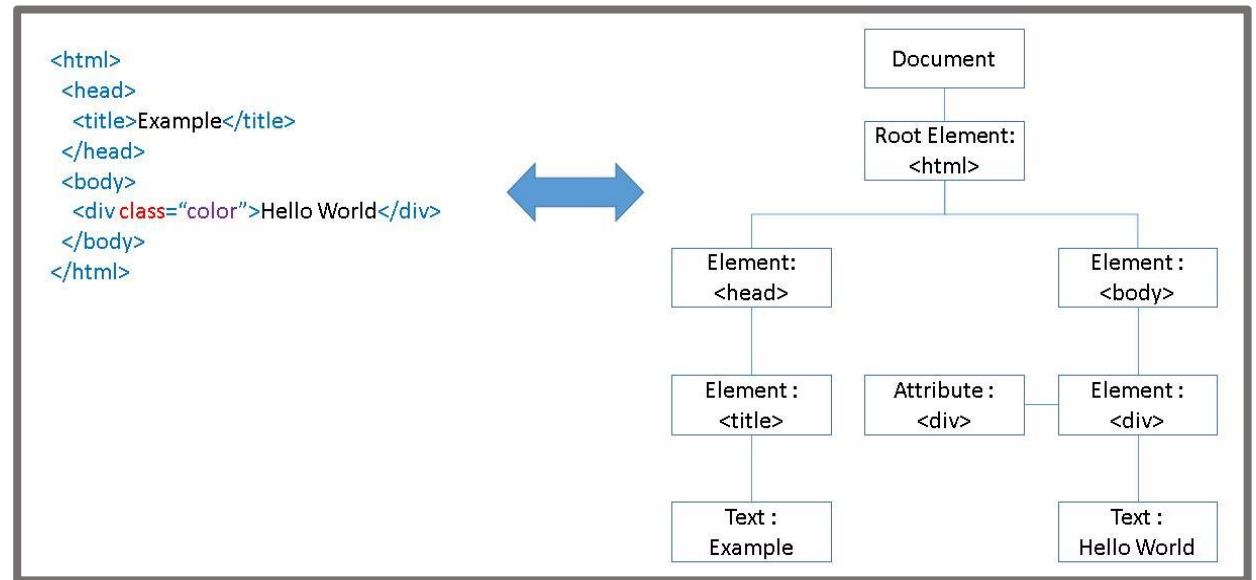
REFUERZO  
JAVASCRIPT

---

# DOM – VIRTUAL DOM

**Document Object Model:** Is the tree of nodes that browsers generate to understand our HTML code.

**Virtual DOM** is a copy that allows us to do heavy calculations without affecting the performance of the real DOM.





# Vue JS

- Progressive Javascript Framework. Can be used for both basic tasks and more complex tasks.
- **Approachable:** HTML, CSS and JavaScript.
- **Versatile:** Incremental, can be scaled between a library and a framework.
- **Performant:** Quick Virtual DOM.



# Create Vue Project

```
✓ Project name: ... <your-project-name>
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit testing? ... No / Yes
✓ Add Cypress for both Unit and End-to-End testing? ... No / Yes
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
```

```
Scaffolding project in ./<your-project-name>...
Done.
```

---

**\* npm init vue@latest**

---

**> npm install -g @vue/cli**

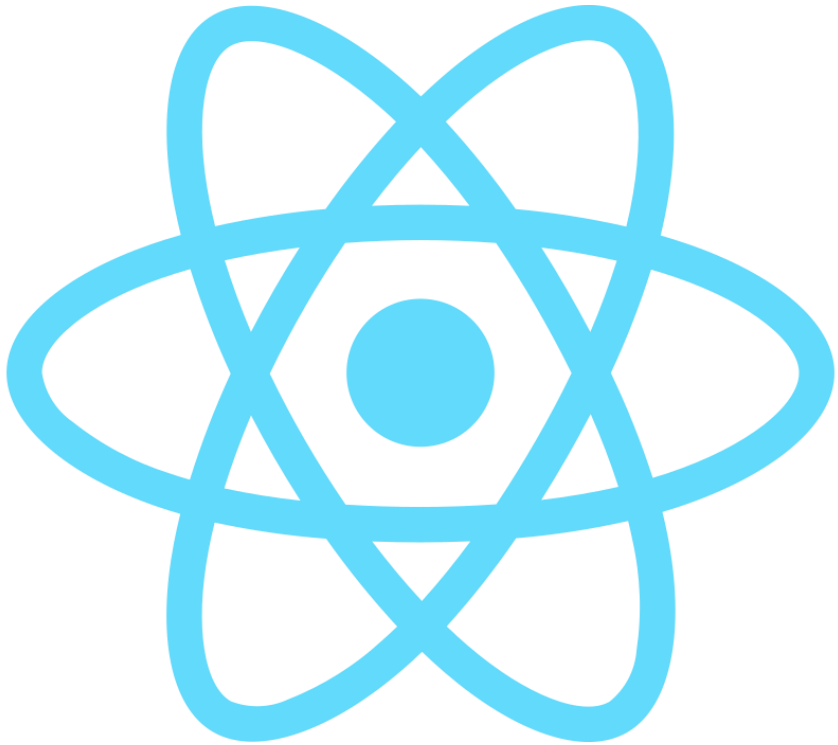
---

**> vue create my-project**

---

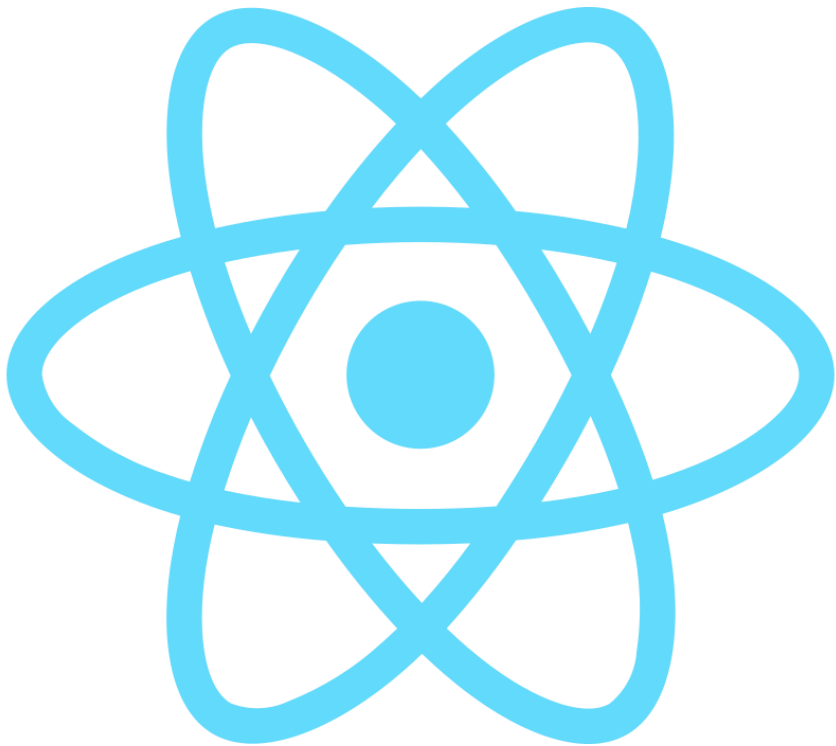
`<script src="https://unpkg.com/vue@next"></script>`

# React JS



- A JavaScript library for building user interfaces. Makes it painless to create interactive UIs.
- **Declarative:** Handles states in your application. React will efficiently update and render just the right components when your data changes.
- **Component-Based:** Build encapsulated components that manage their own state, then compose them to make complex UIs.
- **Learn Once, Write Anywhere:** It doesn't matter about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

# Create React Project



---

**`npx create-react-app my-app`**

---

```
<script
  crossorigin
  src="https://unpkg.com/react@16/umd/react.production.min.js">
</script>
```

```
<script
  crossorigin
  src="https://unpkg.com/react-dom@16/umd/react-dom.production.min.js">
</script>
```

```
<script
  src="https://unpkg.com/babel-standalone@6/babel.min.js">
</script>
```

---



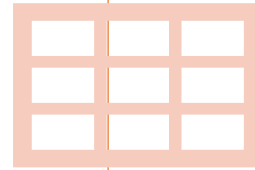
# Angular



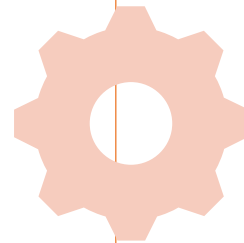
- Angular is an application design framework and development platform for creating efficient and sophisticated single-page apps.
- A component-based framework for building scalable web applications
- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more
- A suite of developer tools to help you develop, build, test, and update your code



# Create Angular Project



```
npm install -g @angular/cli
```



```
ng new my-app
```

# Var – Let – Const

---

	Scope	Redeclare	Reassigned	Hoisted
Var	Global Local	YES	YES	YES
Let	Block	NO	YES	NO
Const	Block	NO	NO	NO

# Var – Let – Const

```
var nombre = 'Jonathan'  
const apellido = 'Lopez'  
let edad = 34
```

```
if ( 1 == 1 ) {  
  var nombre = 'Pedro'  
  const apellido = 'Perez'  
  let edad = 10  
  console.log(nombre, apellido, edad)  
}
```

```
console.log(nombre, apellido, edad)
```

```
var nombre = 'Jonathan'  
const apellido = 'Lopez'  
let edad = 34
```

```
var nombre = 'Pedro'  
const apellido = 'Perez'  
let edad = 10  
console.log(nombre, apellido, edad)
```

```
var nombre = 'Jonathan'  
const apellido = 'Lopez'  
let edad = 34
```

```
nombre = 'Pedro'  
apellido = 'Perez'  
edad = 10
```

```
console.log(nombre, apellido, edad)
```

```
console.log(nombre, apellido, edad)
```

```
var nombre = 'Jonathan'  
const apellido = 'Lopez'  
let edad = 34
```

```
const apellido = 'Lopez'  
let edad = 34
```

```
console.log(nombre, apellido, edad)
```

```
var nombre = 'Jonathan'
```



# Template Strings

---

Are Strings delimited with backtick (``) characters, allowing for multi-line strings, and string interpolation with embedded expressions.

```
const myString = `
  Hello
  World
`

console.log( myString )
```

```
const collegeType = 'Universidad'
const collegeName = 'Autonoma'


console.log( `${ collegeType } ${ collegeName }`)
```




# Objects

- Is a collection of properties.
- A property is an association between a name (or key) and a value.
- A property's value can be a function, in which case the property is known as a method.

```
const myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;
```



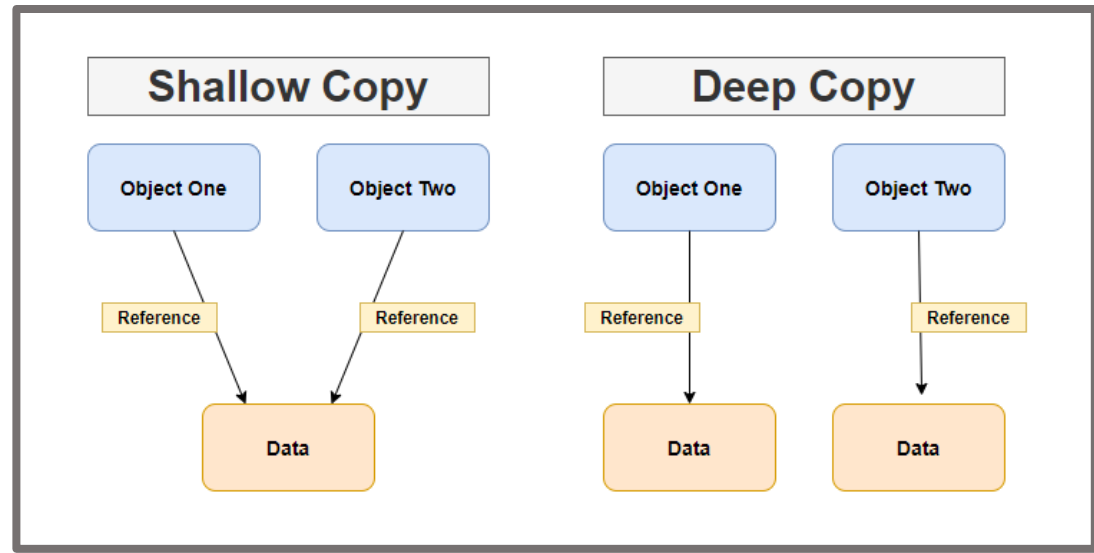
```
const myCar = {  
  make: 'Ford',  
  model: 'Mustang',  
  year: 1969,  
  city: {  
    name: 'Colombia',  
    latLng: {  
      lat: 123,  
      lng: 456  
    }  
  }  
};
```



# SHALLOW COPY VS DEEP COPY

---

```
const myArray = [1, 2, 3, 4];  
myArray.push(5)  
  
const mySecondArray = myArray;  
mySecondArray.push(6)  
  
console.log(myArray, mySecondArray)
```





# Arrays

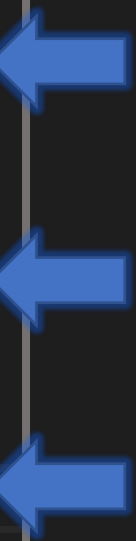
In JavaScript, arrays aren't primitives but are instead objects with the following core characteristics:

- Are resizable and can contain a mix of different data types.
- Are not associative arrays and so, array elements cannot be accessed using arbitrary strings as indexes but must be accessed using nonnegative integers as indexes.
- Are zero-indexed: the first element of an array is at index 0, the second is at index 1, and so on — and the last element is at the value of the array's length property minus 1.
- Array-copy operations create shallow copies

```
const myArray = [1, 2, 3, 4];  
myArray.push(5)
```

```
const mySecondArray = myArray;  
mySecondArray.push(6)
```

```
console.log(myArray, mySecondArray)
```







# Arrays Functions

---

```
▼ [] ⓘ  
  length: 0  
  ▼ [[Prototype]]: Array(0)  
    ▶ at: f at()  
    ▶ concat: f concat()  
    ▶ constructor: f Array()  
    ▶ copyWithin: f copyWithin()  
    ▶ entries: f entries()  
    ▶ every: f every()  
    ▶ fill: f fill()  
    ▶ filter: f filter()  
    ▶ find: f find()  
    ▶ findIndex: f findIndex()  
    ▶ findLast: f findLast()  
    ▶ findLastIndex: f findLastIndex()  
    ▶ flat: f flat()  
    ▶ flatMap: f flatMap()  
    ▶ forEach: f forEach()  
    ▶ includes: f includes()  
    ▶ indexOf: f indexOf()  
    ▶ join: f join()  
    ▶ keys: f keys()  
    ▶ lastIndexOf: f lastIndexOf()  
    length: 0  
    ▶ map: f map()  
    ▶ pop: f pop()  
    ▶ push: f push()  
    ▶ reduce: f reduce()  
    ▶ reduceRight: f reduceRight()  
    ▶ reverse: f reverse()  
    ▶ shift: f shift()  
    ▶ slice: f slice()  
    ▶ some: f some()  
    ▶ sort: f sort()  
    ▶ splice: f splice()  
    ▶ toLocaleString: f toLocaleString()  
    ▶ toString: f toString()  
    ▶ unshift: f unshift()  
    ▶ values: f values()
```



# Arrays Functions

```
const myArray = [1, 2, 3];  
const myArray2 = [4, 5, 6];
```

```
myArray.push(4)  
console.log(myArray)
```

```
const myArray = [1, 2, 3];  
const myArray2 = [4, 5, 6];
```

```
const lastItem = myArray.pop()  
console.log(lastItem)
```

```
const myArray = [1, 2, 3];  
const myArray2 = [4, 5, 6];
```

```
const myArray3 = myArray.concat( myArray2 );  
console.log( myArray3 )
```

```
const myArray = [1, 2, 3];  
const myArray2 = [4, 5, 6];
```

```
const last = myArray.findLast( x => x < 2 )  
console.log(last)
```

```
const myArray = [1, 2, 3];  
const myArray2 = [4, 5, 6];
```

```
const areEvery = myArray.every( x => x < 4 )  
console.log(areEvery)
```



# Spread Operator

- In an object literal, the spread operator (...) enumerates the properties of an object and adds the key-value pairs to the object being created.
- It creates deep copies

```
const myCar = {  
  make: 'Ford',  
  model: 'Mustang',  
  year: 1969,  
  city: {  
    name: 'Colombia',  
    latLng: {  
      lat: 123,  
      lng: 456  
    }  
  }  
};
```

➡ `const mySecondCar = myCar;`  
`mySecondCar.make = 'Toyota'`

➡ `console.log(myCar, mySecondCar)`

```
const myCar = {  
  make: 'Ford',  
  model: 'Mustang',  
  year: 1969,  
  city: {  
    name: 'Colombia',  
    latLng: {  
      lat: 123,  
      lng: 456  
    }  
  }  
};
```

➡ `const mySecondCar = { ...myCar };`  
`mySecondCar.make = 'Toyota'`

➡ `console.log(myCar, mySecondCar)`

# Homework

---

- ✓ Research and use all array functions.
- ✓ Create a new project in **GIT** called **Homeworks**
- ✓ Create a new branch called **01 – Arrays**
- ✓ Upload homework to that branch





## DESTRUCTURING ASSIGNMENT

JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

```
const [a, b] = [10, 20]
```

```
const {name: university, city} = { name: 'Autonoma', city: 'Cali' }
```

```
console.log(a)
```

```
console.log(university)
```





# Functions

A set of statements that performs a task or calculates a value, but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output

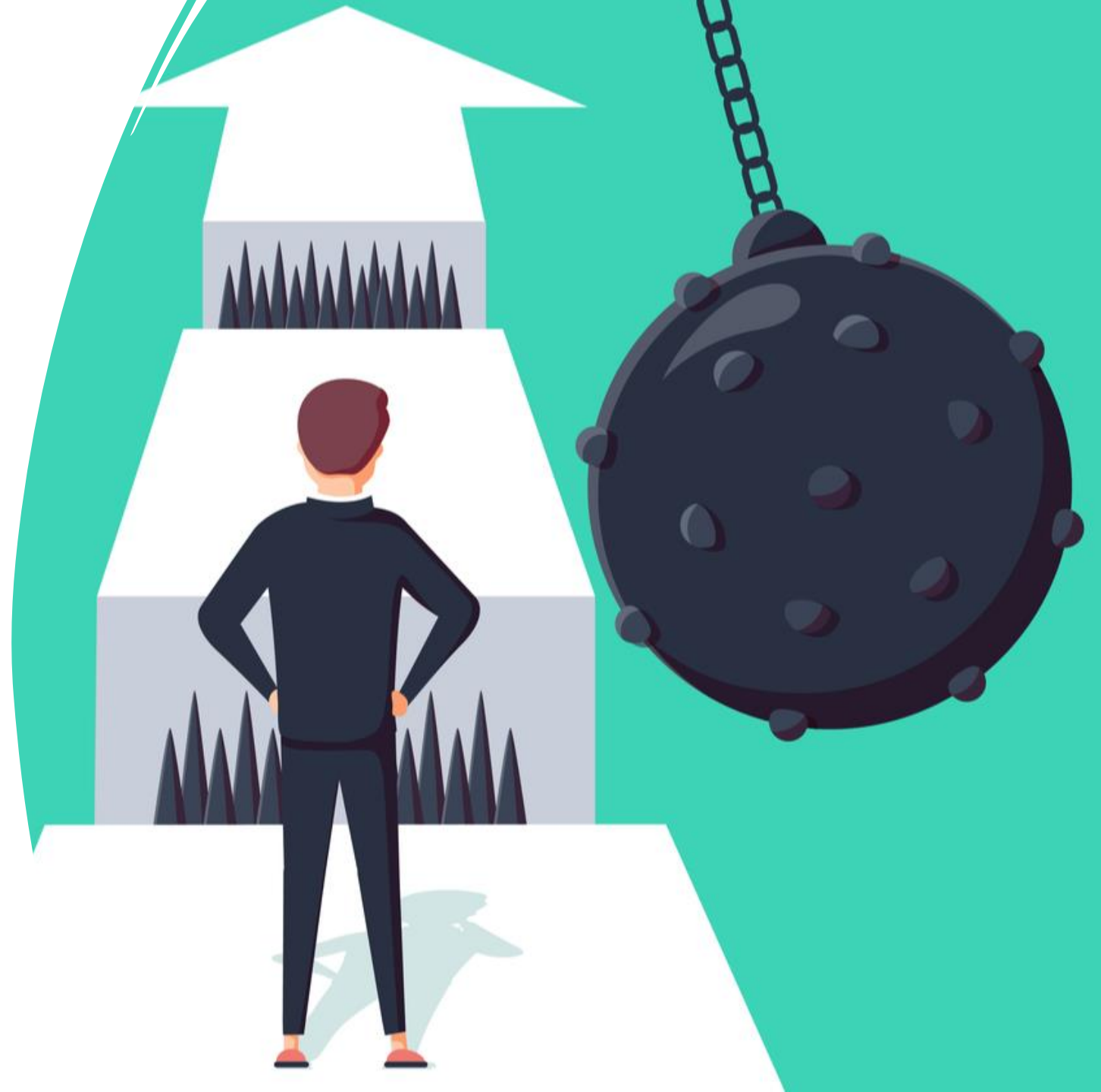
```
function oldRegularFunction (a, b) {  
    return a + b  
}  
  
const newRegularFunction = function (a, b) {  
    return a + b  
}  
  
const arrowFunction = (a, b) => {  
    return a + b  
}  
  
console.log( oldRegularFunction(2, 3) )  
console.log( newRegularFunction(2, 3) )  
console.log( arrowFunction(2, 3) )
```



# CHALLENGE

---

- ✓ Research difference between Arrow Functions and Regular Functions
- ✓ Create a new function in Regular and Arrow types, which should receive a number and will print in console if that number is either odd or even





# Imports and Exports

---

## Import


- Is used to import read-only live bindings which are exported by another module.
- The imported bindings are called live bindings because they are updated by the module that exported the binding but cannot be modified by the importing module.

## Export

- Is used to export values from a JavaScript module.
- Exported values can then be imported into other programs with the import declaration or dynamic import. The value of an imported binding is subject to change in the module that exports.




```
import { animals } from "../animals";  
  
console.log( animals )
```



# Imports and Exports

---

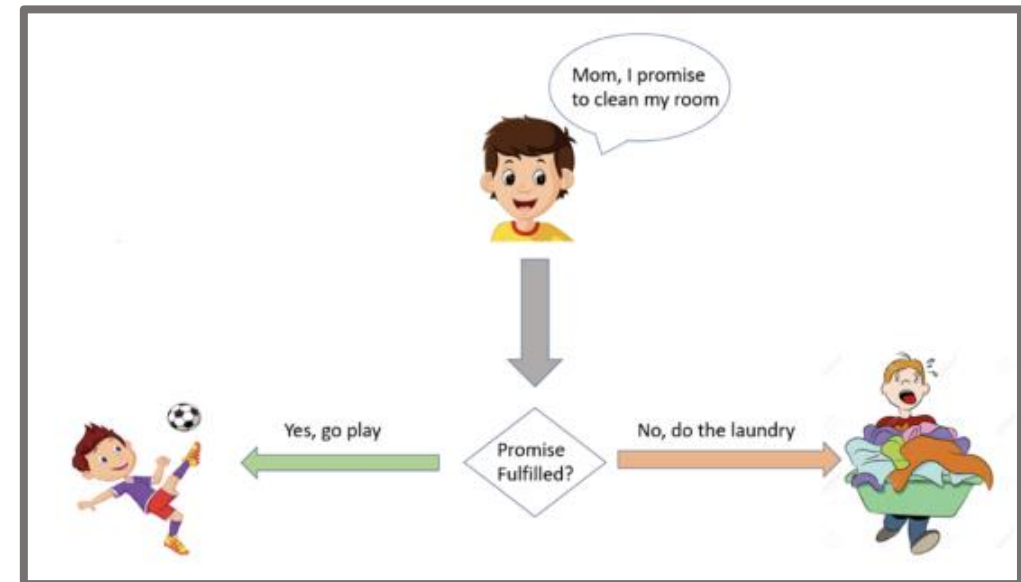
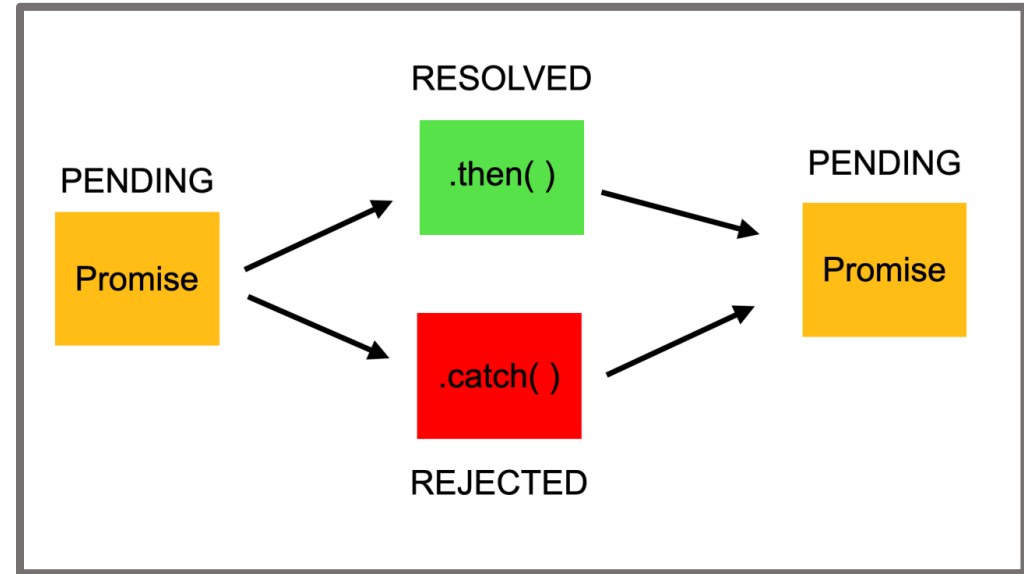
```
// animals.js  
export const animals = [  
  {  
    breed: 'Canine',  
    name: 'Angel'  
  },  
  {  
    breed: 'Canine',  
    name: 'Angel'  
  },  
  {  
    breed: 'Feline',  
    name: 'Orion'  
  },  
  {  
    breed: 'Feline',  
    name: 'Luna'  
  },  
  {  
    breed: 'Bird',  
    name: 'Chicken'  
  },  
  {  
    breed: 'Bird',  
    name: 'Toucan'  
  },  
]
```






# Promises


Represents the eventual completion (or failure) of an asynchronous operation and its resulting value.





```
const promesa = new Promise( (resolve, reject) => {  
  /// TODO  
  resolve(); // if promise is OK  
  reject(); ; // if promise is Fails  
});
```





```
promesa.then( (data) => {  
  console.log(data)  
}).catch( err => {  
  console.log( err )  
})
```



```
const getPromesa = () => {  
  return new Promise( (resolve, reject) => {  
    resolve('Hello World');  
  })  
}  
  
getPromesa()  
  .then( data => {  
    console.log(data)  
  })  
  .catch( error => {  
    console.error( error)  
  });
```



```
const getPromesa = () => {  
  return new Promise( (resolve, reject) => {  
    reject('Internal Error');  
  })  
}  
  
getPromesa()  
  .then( data => {  
    console.log(data)  
  })  
  .catch( error => {  
    console.error( error)  
  });
```



# Promises

---



# Async - Await

Is a function declared with the `async` keyword, and the `await` keyword is permitted within it.

The `async` and `await` keywords enable asynchronous, promise-based behavior to be written in a cleaner style, avoiding the need to explicitly configure promise chains.

```
const getImagen = async () => {  
  try {  
    const apiKey = 'Ns4NdhIX2lHEqu9xdgsozNuiNyuy937q';  
    const resp = await fetch(`http://api.giphy.com/v1/gifs/random?api_key=${ apiKey }`);  
    const { data } = await resp.json();  
    return data  
  } catch(error) {  
    console.error( error )  
  }  
}  
  
getImagen();
```



# Async – Await vs Promise

---

```
const petition = new Promise( (resolve, reject) => {  
  resolve ( fetch('https://api2.binance.com/api/v3/ticker/24hr') )  
});  
  
petition  
  .then( resp => resp.json())  
  .then( (data) => {  
    console.log( data );  
  })  
  .catch( error => console.log('Error:', error) )
```

```
const petition = async() => {  
  try {  
    const resp = await fetch('https://api2.binance.com/api/v3/ticker/24hr');  
    const data = await resp.json();  
    console.log( data );  
  } catch (error) {  
    console.log( error );  
  }  
}  
  
petition2()
```

```

switch ( condition ) {
  case vaue1:
    // TODO
    break;
  case vaue2:
    // TODO
    break;
  default:
    // TODO
    break;
}

```

```

if ( condition ) {
  // TODO
}

```



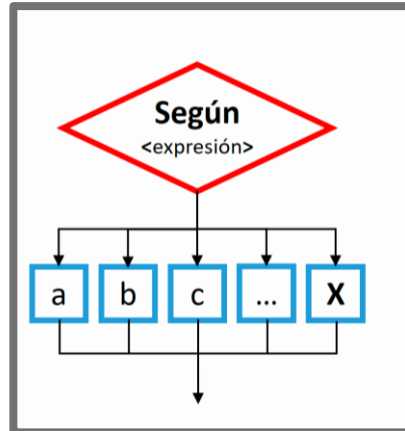
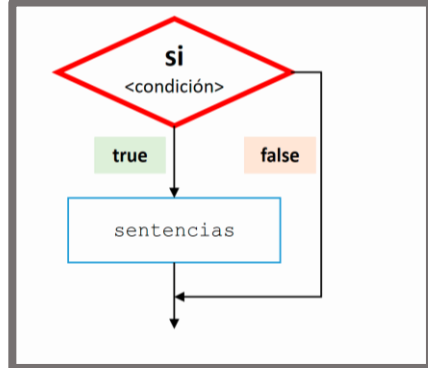
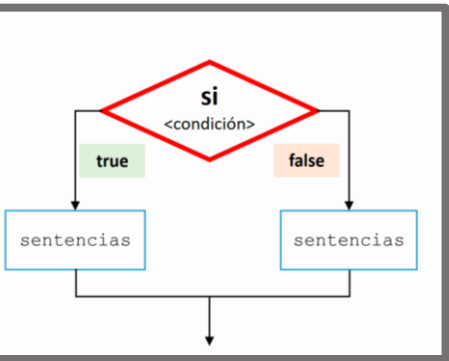
# Selective Structures

- They are used to make logical decisions.
- A condition is evaluated and depending on its result, one option or another is made

```

if ( condition ) {
  // TODO
} else {
  // TODO
}

```

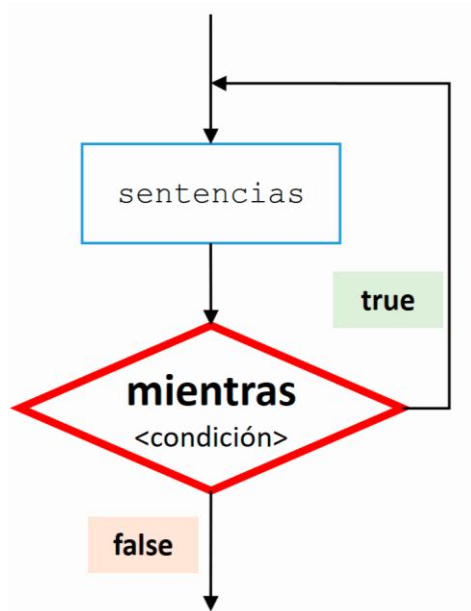




# Null Check Ternary

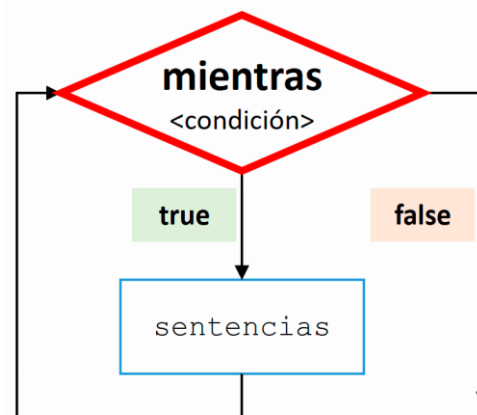
---

```
const myObject = {  
  key: "first"  
}  
  
let result = false;  
if ( myObject.key2 ) {  
  result = true;  
}  
  
result = myObject.key2 ? true : false;  
  
const result2 = myObject.key2 || 'Noting';  
  
const result3 = myObject.latLng?.lat || 'Empty';  
  
console.log(result, result2, result3)
```



```
do {  
    // TODO  
} while( condition )
```

```
while( condition ) {  
    // TODO  
}
```



# Loops

They are used to repeat same instructions several times, until the assigned condition is done.



# Loops

They are used to repeat same instructions several times, until the assigned condition is done.

```
const myObject = { a: 1, b: 2, c: 3 };  
  
for (const key in object) {  
  console.log(`${ key } : ${ myObject[key] }`);  
}
```

```
for ( init; condition ; increment ) {  
  // TODO  
}
```

