JavaScript

Javascript is a Scripting Language that is initially used for Client Side in Web Development along with html.

JavaScript ("JS" for short) is a full-fledged dynamic programming language

It was invented by Brendan Eich (co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation).

JavaScript was invented by Brendan Eich in 1995, and became an ECMA standard in 1997.

ECMA-262 is the official name of the standard. ECMAScript is the official name of the language

But now it can be used for almost all type of development work

- Web Dev (Front End, BackEnd, Data Layer)
- Desktop App Dev
- Mobile App Dev
- Game Dev etc..

JavaScript Runtimes

Runtime basically means how we are going to run our Code in CPU.Which interpretors/SW used to Convert our code to Machine Code.

To Execute JavaScript Code, A Runtime is needed.

In case of Browsers there are inbuilt JS Engines that executes JS. Popular JS Engines are V8 in Chrome, Spider Monkey in Firefox etc..

Other than browsers there are **Node** and **Deno** that can be used as JS Runtime Environments.

Node is the most opular for now for Non browser JS Environment.

What JS can and cannot Do in Browser

- JS can be used to Manipulate Web Page
- JS can be used respond user's action on page.
- JS can be used to Interact with SERVER.
- JS can be used to Validate Data given by Users before sending it server.
- JS cannot access Users OS and File System.

- JS cannot access media devices such as Camera and Mic without Users consent/Permission.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

The Above Code will prompt Hello World in the Browser.

The browser runs the HTML code Line wise in a Tree like Structure.


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```


DOM - Document Object Model

Parsing - means the DOM tree is being made in the Background.

Now once the Tree is being made after **Parsing** we need to Paint it in the Browser. This is called **Rendering**.

The Code in the Script Tag will Stop/**Block** the Rendering (running) of the Lines Below it. First all the commands in the Script tags will run then only we move to the next child.

If we Want our JavaScript Code To run from external file.


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```


Difference Between Defer and Async and Normal

In **Defer** the HTML code will Parsed first without running Script(JS) file. and once the HTML code is parsed it'll execute the JS code.

In **Async** the HTML code will be parsed only till the time JS file is being Downloaded parallely means(JS needs its files to download first and then runs on the browser). and when the file is download it'll block the HTML and runs the Script.

- Async doesnt give gurateed of the Order of sripts to be run. **First downloads First runs**.

In **Normal Js** the HTML code will be blocked , and first the Script file is executed after that it moves to HTML code

Normal Example

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

In Normal - Runs Order wise Blocks HTML

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Defer - Runs HTML code then Runs Scripts orderwise

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Async - Executes Scripts which are downloaded first (No Gurantee of Order)

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Variables and DataType in JavaScript


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```


JavaScript is Dynamic Typed language


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
 alert("2 plus 2 equals " + 2 + 2);
...JavaScript automatically converts the numbers to strings, and
displays the message "2
plus 2 equals 22".
```

Operators

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Ways to take Input from Users

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

How to Access and Manipulate DOM in JS

This is how we access element in Java Script

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

If we use Class element to get then we'll get

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Events and Event Handling

It means when the user Interacts with our Web Page some Actions takes place and the Actions that has been occured is known as Events.

```
Element(Action by User)--> Event (objects) --> Event listener Function
(Calls that particular function)
```

```
Inline event-handling means that you combine bits of JavaScript with
HTML markup.
Here's a line that displays a link and then displays an alert when the
user clicks it.
```

```
<a href="#" onClick="alert('Hi');">Click</a>
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

If we want function to run whenever the Button gets clicked

Event == button **Event Listener** == function fun()

Different ways to call Event Listener:

onclick

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

We can add multipe functions in Onclick

```
<button id='btn1' onclick="fun();fun2();fun3()">click 1</button>
```

The Above function will be called only when the button (Action) gets Performed.

Second way

```
function Student(name, age)
{
        this.name=name;
        this.age=age;
        this.print= function (){
            console.log("name ="+ this.name);
            console.log("age ="+ this.age);
        }
}

// var s1 = {
//     name:"rahul",
//     age:45
// }

// var s2= {
//     name:"ram",
//     age:34
// }


var s1 = new Student("rahul", 45);
var s2 = new Student("ram", 34);

s1.print();
s2.print();
```

In the above example if We add two or more functions then Only the Last function will Execute.

Third way

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

In this Both EventListener will execute

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

JavaScript Identifiers are unique names.In JavaScript, identifiers are used to name variables , keywords, functions,labels).

JavaScript variables are containers for storing data values. must have unique names

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
onblur = fun()

onfocus = fun()


GOTO JAVASCRIPT Event List
```

Let

Variables defined with let cannot be Redeclared.

Variables defined with let must be Declared before use.

Variables defined with let have Block Scope.

Const

Variables defined with const cannot be Redeclared.

Variables defined with const cannot be Reassigned.

Variables defined with const have Block Scope.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

forms in html

- **action** - takes data from form to path to Resource
- **method** - get/post
- **enctype** - If we want files in the FORM like (docs,img etc).

Get - we can see the data in the **URL**

Post - we can't see the data in the POST

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Validation inside the FORM

We want data to be validated before sending unnecessarily to the SERVER, as server will be filled with Garbage.

```
function Student(name, age)
{
        this.name=name;
        this.age=age;
        this.print= function (){
            console.log("name ="+ this.name);
            console.log("age ="+ this.age);
        }
}

// var s1 = {
//     name:"rahul",
//     age:45
// }

// var s2= {
//     name:"ram",
//     age:34
// }



var s1 = new Student("rahul", 45);
var s2 = new Student("ram", 34);

s1.print();
s2.print();
```

The Above form will trigger its ONSUBMIT event and checl if the form VALIDATION is correct or NOT

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Event Bubbling

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

The Above Functions will run heirarchically

String

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Finding Segment

If the segment exists, the method finds the index of the first character of the segment and assigns it to the variable firstChar. If the segment doesn't exist, the method assigns -1 to the variable, so you know it's not there.

Now we can replace the banned phrase with the preferred phrase with less coding.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Note: The indexOf method can only identify the character at a particular location. It can't

change the character at a location.

Strings: Finding a character at a location

The following code finds the last character in the string.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Strings: Replacing characters

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

String.trim()

The trim() method removes whitespace from both sides of a string:

All string methods return a new string. They don't modify the original string. Formally said: Strings are immutable: Strings cannot be changed, only replaced.


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
  scoreAvg = Math.round(scoreAvg);
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Slicing in JS

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Converting strings to integers and decimals


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Array in JavaScript

Array is the collection of similar type of data items stored at contiguous memory locations.

Arrays are lists of ordered, stored data. They can hold items that are of any data type. Arrays are created by using square brackets, with individual elements separated by commas.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
arr.splice()

arr.concat(arr2)
```

Function

A function is a reusable set of statements to perform a task or calculate a value. Functions can be passed one or more values and can return a value at the end of their execution. ***


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```


Anonymous Functions

Anonymous functions in JavaScript do not have a name property. They can be defined using the function keyword, or as an arrow function. See the code example for the difference between a named function and an anonymous function.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Arrow function

The syntax for an arrow function expression does not require the function keyword and uses a fat arrow => to separate the parameter(s) from the body. However, they are limited and can't be used in all situations.

There are several variations of arrow functions:

Arrow functions with a single parameter do not require () around the parameter list.
Arrow functions with a single expression can use the concise function body which returns the result of the expression without the return keyword. ***

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Limitations
Arrow functions do not have their own bindings to this or super, and should not be used as methods.
Arrow functions cannot be used as constructors.
Arrow functions cannot use yield, within its body

Callback in JS is like Decorators in Python

Callback functions are functions that are passed as arguments in other functions. A callback function can then be invoked during the execution of that higher order function (that it is an argument of). ***

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

In the previous Slides we saw different ways to define Functions in JavaScript.

Let's recall :-

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Array Sorting - Array .sort() methods assumes every elements to be string.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

By Default array.sort() will sort stringwise ****

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

We use a helper function to sort heterogeneous or Numeric elements.

the helper function will return True(+ve) when a is greater but False(-ve) when smaller.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

What if we want to sort the above array.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

The above function will sort AGE wise *****

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Shuffling method

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Array Iterations

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Map will store Values in a New Array hence we need to Catch new Values to a New Array. The callback function Must have return Statement to get values innside MAP.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Array.map()

The map() method creates a new array by performing a **function** on each element of the array.

The map() method does not execute the function for array elements without values.

The map() method does not change the original array.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

***

filter( )

- filter will return only values which are True
- filter will return all the values at once

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Array.findIndex()

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

---

json

json is a format which is udes to STORE multiple DATA of single ENTITY.

```
var person = { "name" : "rahul",
               "age" : 23,
```

```
  "marks" : 70
}
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

****

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Use SQUARE NOTATION [ ].

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

The Above CODE will not work as **our dot(.) notation** will search for the value **p** which is not present in the json.

Hence we will use **SQUARE** method to access "age".

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Nested Json

To Access NESTED JSON objects remember below syntax.

We use [ ] square notation to access array elements and dot(.) notation to access JSON elements

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Advanced Operators

- new -> used to create new object.
- () grouping
- In
- of
- rest
- spread
- type of
- instance of.


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
typeof() - only tells primitive ( int float array ..)type data types
           for Non primitive returns Object.
```

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by JavaScript (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

```
instanceof() -  operators tells us whether the object is the
constructor of that class or not.
                It always returns boolean values


The Left Hand Side (LHS) operand is the actual object being tested to
the Right Hand Side (RHS) operand which is theactual constructor of a
class.
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
spread operator uses tiple dots (...arr).
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

---

rest operator - if we dont know how many values we will get from the array to the function.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

***

Constructor -(a function) which is used to create JSON object dynamically. here we use this reference .

```
function Student(name, age)
{
        this.name=name;
        this.age=age;
        this.print= function (){
            console.log("name ="+ this.name);
            console.log("age ="+ this.age);
        }
}

// var s1 = {
//     name:"rahul",
//     age:45
// }

// var s2= {
//     name:"ram",
//     age:34
// }



var s1 = new Student("rahul", 45);
var s2 = new Student("ram", 34);

s1.print();
s2.print();
```

---

here **new** creates a blank JSON object for Student function.

```
function Student(name, age)
{
        this.name=name;
        this.age=age;
        this.print= function (){
            console.log("name ="+ this.name);
            console.log("age ="+ this.age);
        }
}

// var s1 = {
//     name:"rahul",
//     age:45
// }

// var s2= {
//     name:"ram",
//     age:34
// }


var s1 = new Student("rahul", 45);
var s2 = new Student("ram", 34);

s1.print();
s2.print();
```

If there is a function that is same for all the Objects we use a COMMON SHARED memory mechanism such that it'll not use uneccesary memory we use Prototype.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Context Binding

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

No Context Binding

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Manual Context binding - we can use the below fun to manually bind context.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

****

We cannot bind Arrow function like below

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

***

Only in this case Context Binding with Arrow function can be applied

---

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

****

Closure

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Error Handling

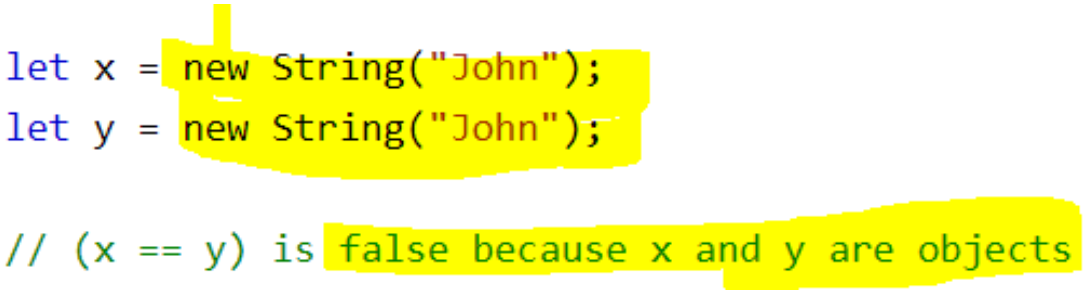
```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

***

Synchronous & Asynchronous

IF we Send dependent CALL in Asynchronous function. Or we are using dependent code in Asynchronous way in the below image



The above situation is called PYRAMID OF DOOM/CALLBACK HELL because it'll be difficult to understand what the functions are doing.

To restrain from above we use PROMISEs.

Callbacks

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

After 3000 is finished callback function will be called that is myFunction

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Functions running in parallel with other functions are called asynchronous

Promise is an object that represents some pending task that can be fulfilled or can be rejected(failed) in near fututre.

- A “producing code” that does something and takes time. For instance, some code that loads the data over a network. That’s a “singer”.
- “consuming code” that wants the result of the “producing code” once it’s ready. Many functions may need that result. These are the “fans”.
- promise is a special JavaScript object that links the “producing code” and the “consuming code” together. In terms of our analogy: this is the “subscription list”. The “producing code” takes whatever time it needs to produce the promised result, and the “promise” makes that result available to all of the subscribed code when it’s ready.
- Promise provides us two functions **then() and catch()**, with the help of them we can attach our handler functions for both the situations.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

So to summarize: the executor runs automatically and attempts to perform a job. When it is finished with the attempt, it calls resolve if it was successful or reject if there was an error.

The promise object returned by the new Promise constructor has these internal properties:

- **state — initially "pending"**, then changes to either "fulfilled" when resolve is called or "rejected" when reject is called.
- **result — initially undefined,** then changes to value when resolve(value) called or error when reject(error) is called.

So the executor eventually moves promise to one of these states: ***


```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects

let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Promise.then() takes two arguments, a callback for success and another for failure. Both are optional, so you can add a callback for success or failure only.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

If we want asynchronous (parallel requests) we use

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

If we want Dependent Asynchronous code i.e, Callback hell

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Promise

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Observable

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Objects

Objects are used to store keyed collections of various data and more complex entities.

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything. ***

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

A variable assigned to an object stores not the object itself, but its "address in memory" – in other words "a reference" to it.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Difference between == and ===

== : checks whether the values are equal or not (it doesnt care about data types)

=== : it checks equality as well as type of the variable.

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Objects cannot be compared

```
let x = new String("John");
let y = new String("John");

// (x == y) is false because x and y are objects
```

Note that comparing two JavaScript objects will always return false.