códigofacilito

Fundamentos de JavaScript

Uriel - CTO de Código Facilito

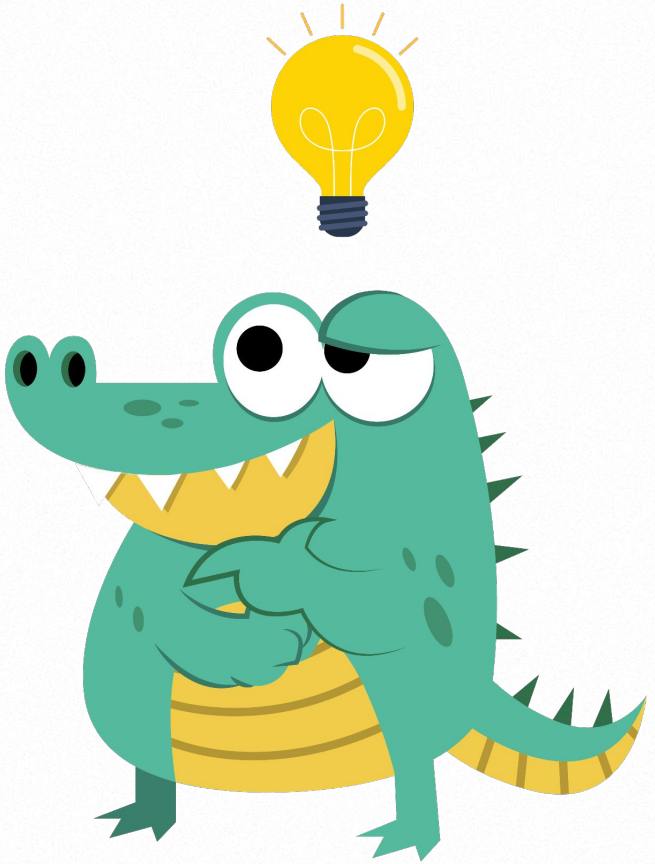
Bienvenidos y Bienvenidas

- Funciones
- Scope
- Contexto
- Valores por valor y referencia
- Promesas

- Módulos
- Otras características de JS
- Conceptos de programación funcional

Introducción a la clase

Vamos a dar un repaso de algunas características de JavaScript que van a ser útiles para seguir las clases de React



Curso profesional de JavaScript

Aprende JavaScript a profundidad, desde las bases del lenguaje, hasta el trabajo con objetos, programación asíncrona, novedad...

codigofacilito.com

Funciones

Veámoslo en práctica:

- Declaración
- Declaración vs Expresión
- Hoisting
- Arrow Functions

En resumen:

- **Una función puede ser “creada” con una declaración de función o una expresión de función**

- **Las declaraciones de función se mueven al principio del alcance, por lo que pueden ser usadas antes de aparecer en el código, a esto se le llama hoisting**

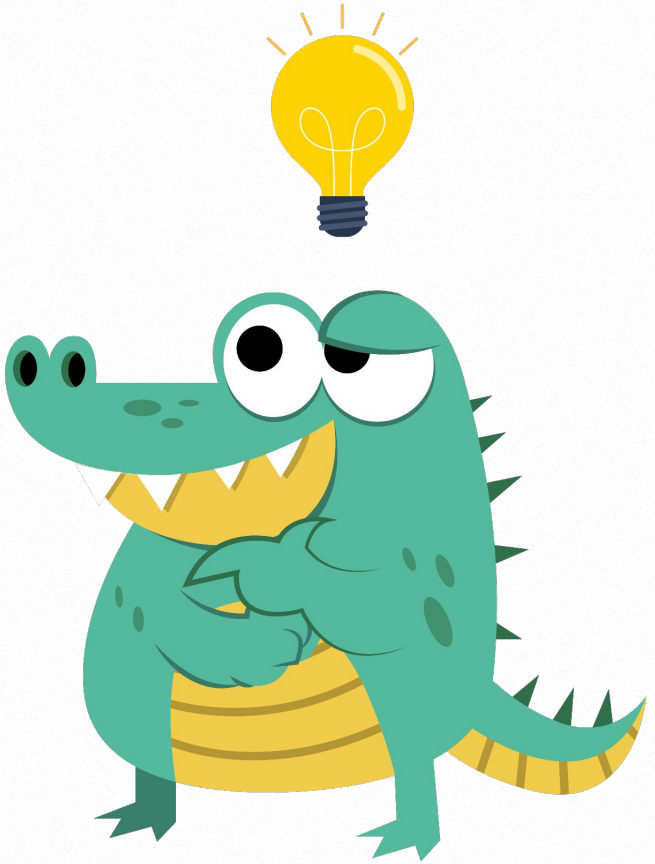
- Las expresiones de función pueden o no tener nombre.

- Las expresiones de función sin nombre, se conocen como funciones anónimas.

- Las expresiones de función pueden usar la sintaxis de flecha

El alcance o scope

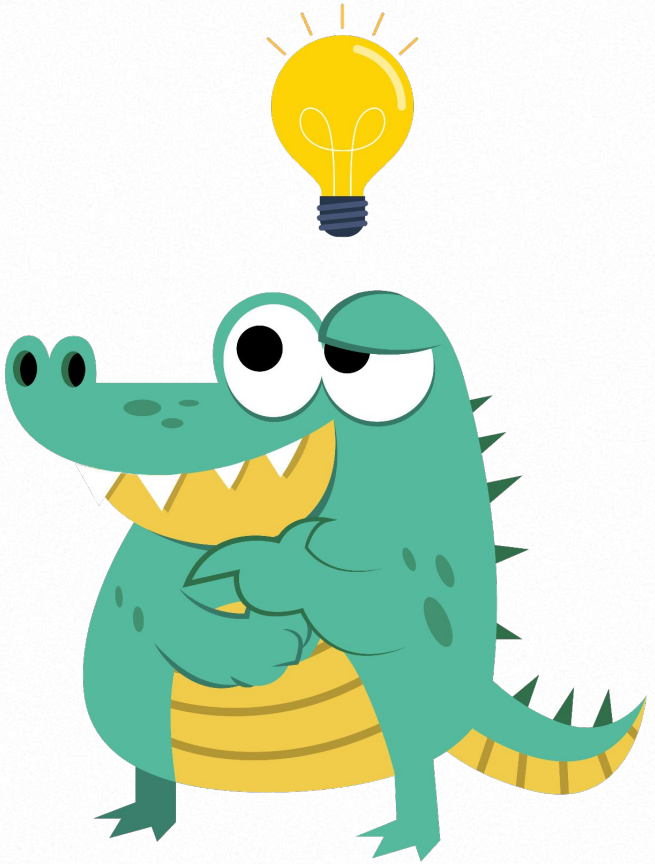
El scope es una colección de variables, funciones y objetos, que están a tu alcance en algún punto de tu código



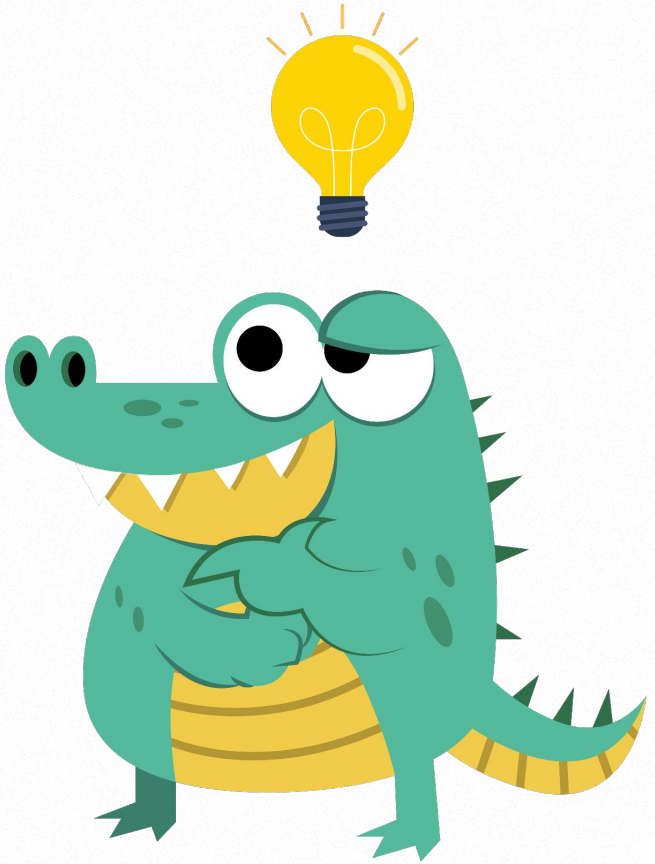
En JS hay dos tipos de alcance

- Scope global
- Scope local

El alcance global es todo aquello que no ha sido declarado dentro de un bloque o una función.



El alcance local hace referencia a todos los elementos disponibles solo para una función. Cada vez que llamas una nueva función, se crea un alcance local para esta función.



Veámoslo en práctica:

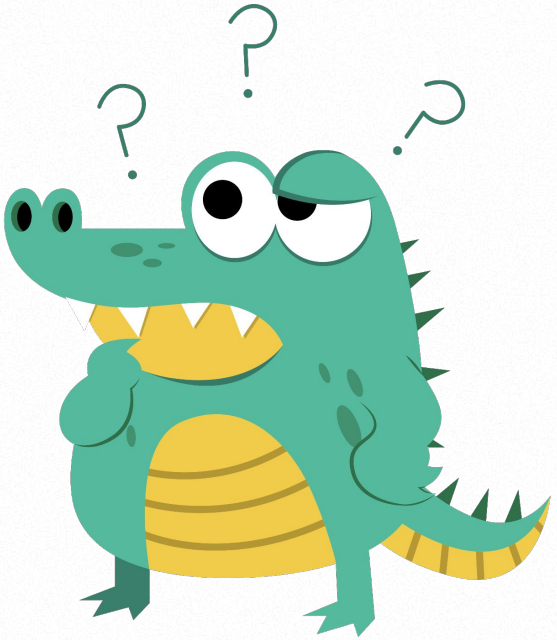
- Variables globales
- Variables locales

Globales	Locales
Se usan en el cuerpo principal	Solo en la función declarada
Se usan en funciones	

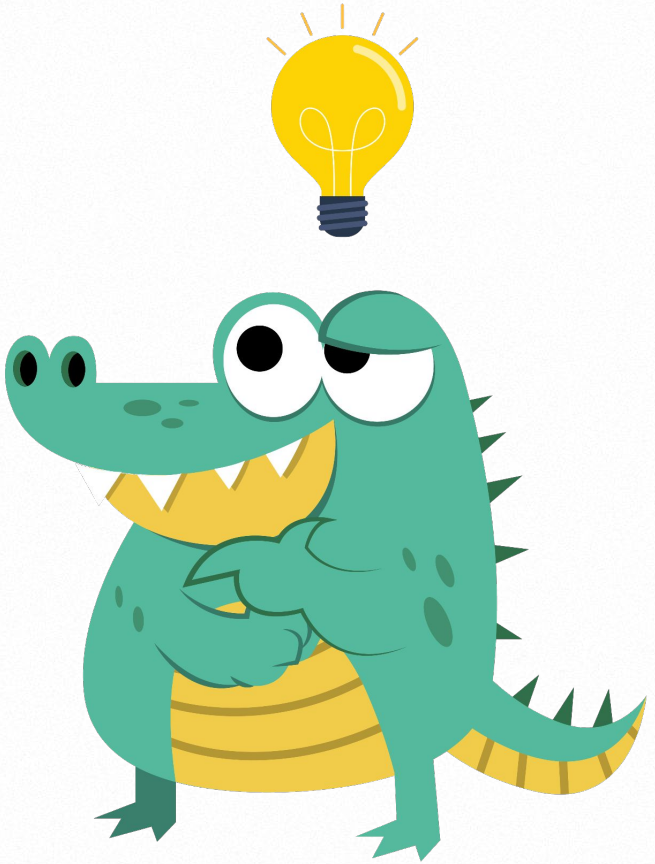

Alcance global
Alcance local


Cada usuario es como una
función, y tiene su propio alcance

¿Cuál es el problema del alcance global en este ejemplo?



Ana función debe operar solo con la información del alcance local, y todo aquello que necesite del exterior debe comunicarse por argumentos, y todo lo que necesita comunicar hacia el exterior debe ser vía el retorno.



En resumen:

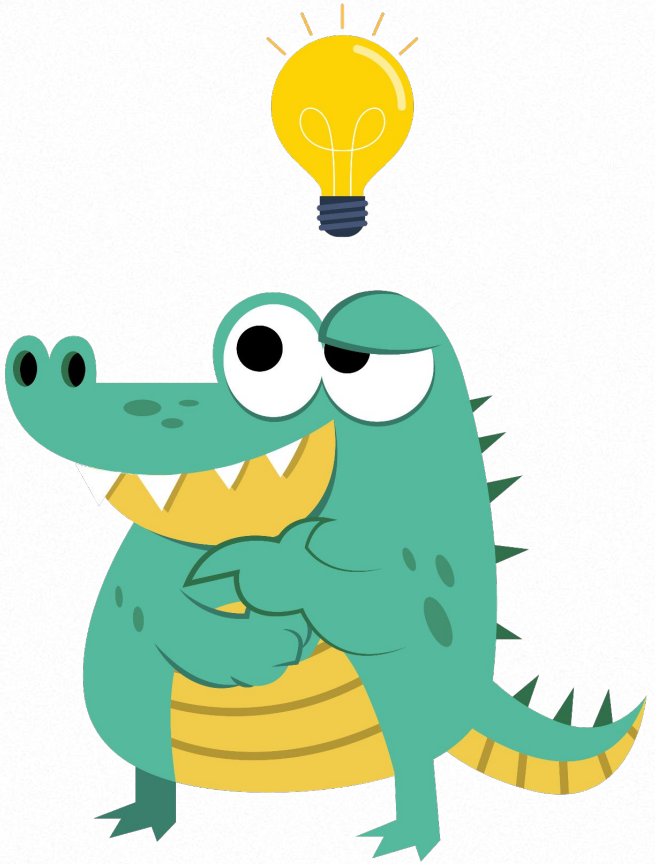
- Tenemos dos tipos de alcance: Local y Global
- El global recuerda que está disponible en todas las partes del código, pero no es recomendable abusar de su uso.

- El local es el que se crea para cada función cuando esta se ejecuta.
- De ser posible,se recomienda que prefieras colocar tus elementos en el scope local de una función y no en el global

El alcance o scope.

El alcance de bloque

En ediciones nuevas de JavaScript se introduce el alcance de bloque.



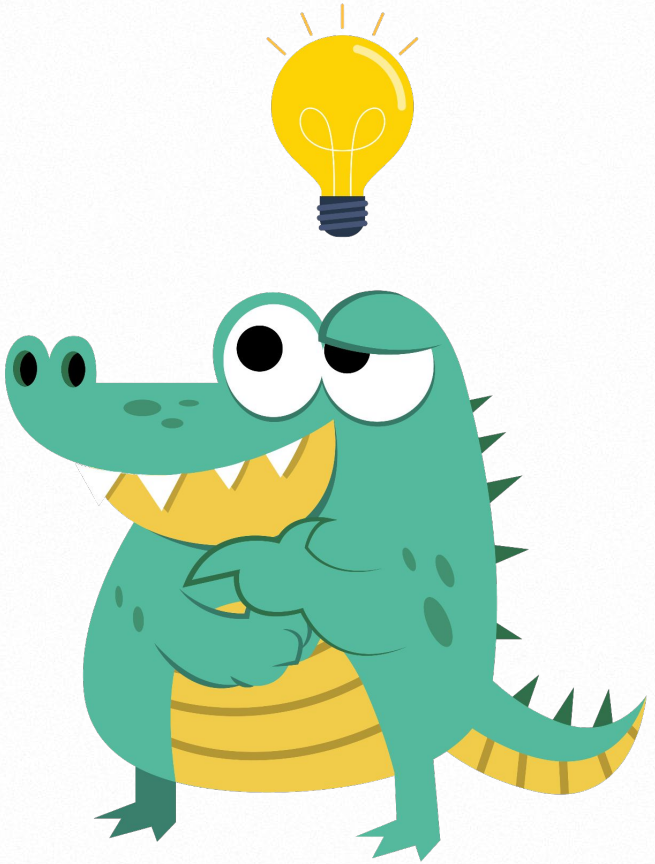
Veámoslo en práctica:

- Alcance de bloque

El alcance o scope.

Hoisting de variables

Hoisting es el concepto por el que, la declaración de funciones y la declaración de variables, se mueven al inicio del alcance.



Veámoslo en práctica:

- Hoisting de variables

En resumen:

- **var siempre se aloja en el scope local dentro de una función, o en el global fuera de una función**

- **let|const puede alojarse en el scope, local, de bloque, y global, según sea declarada en una función, en un bloque, o fuera de ambos.**

- El valor para una variable con var, que no ha sido asignada, es undefined.

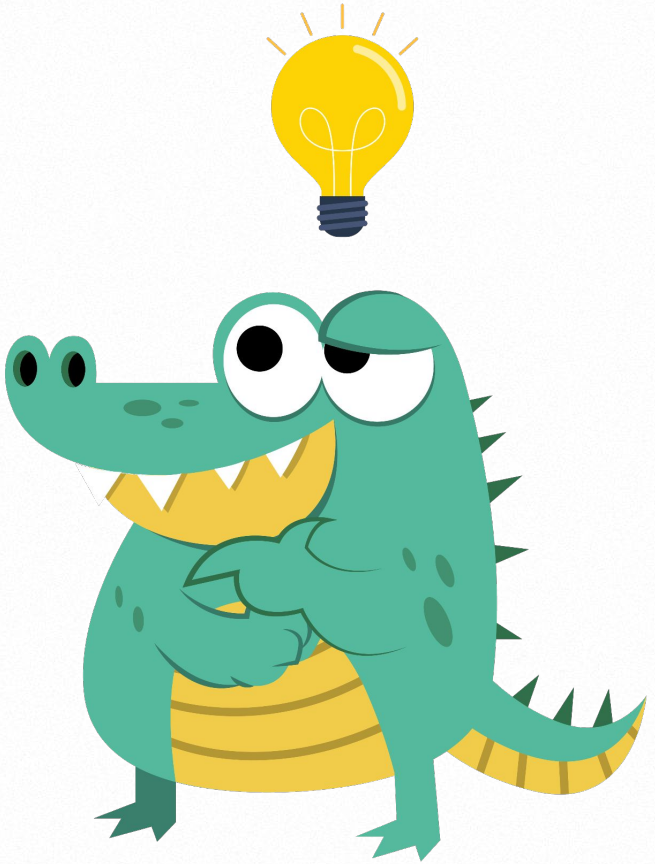
- El valor para una variable con let, que no ha sido asignada, es unitialized.

- Las variables también son alojadas al tope del scope, sin embargo, si intentas usar una variable no inicializada, recibirás un error.

El contexto

El contexto es el valor que tiene this y usualmente hace referencia al objeto sobre el que se está ejecutando el código.

this == contexto



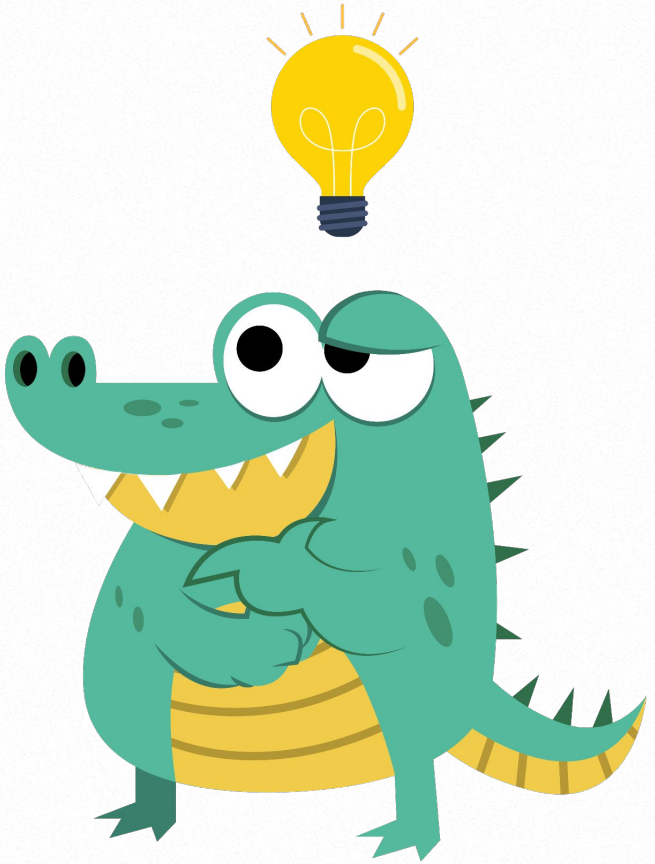
Veámoslo en práctica:

- El valor de this en distintos escenarios

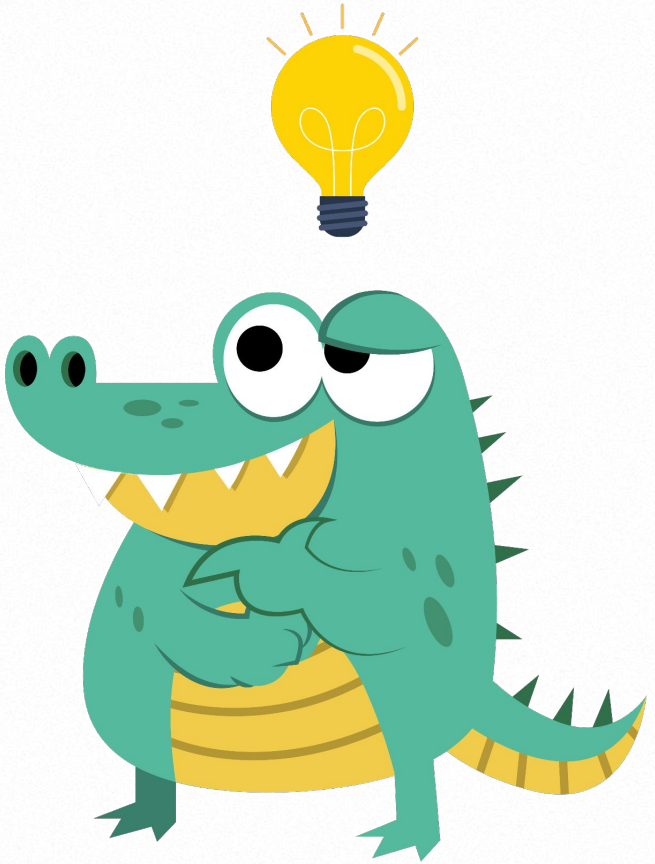
El alcance o scope.

El contexto y las funciones de flecha

Las funciones de flecha no asignan un valor de this, lo heredan del contexto en que se ejecutan.



Como se menciona en esta referencia, si usas this dentro de una función de flecha, este se toma del exterior.



Veámoslo en práctica:

- El valor de this en las funciones de flecha

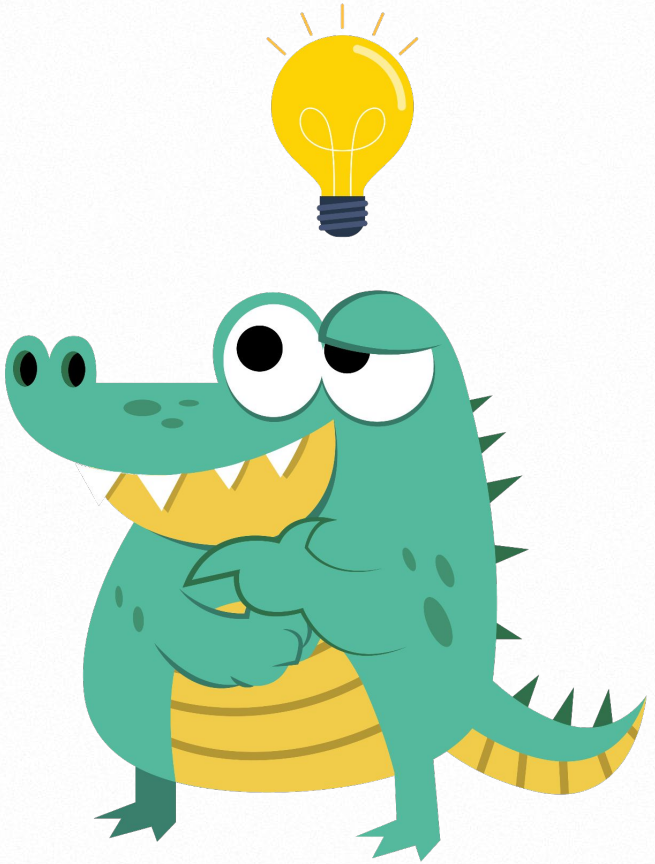
En resumen:

- **Las funciones de flecha no cambian el valor de this, lo heredan o lo toman de “afuera”.**

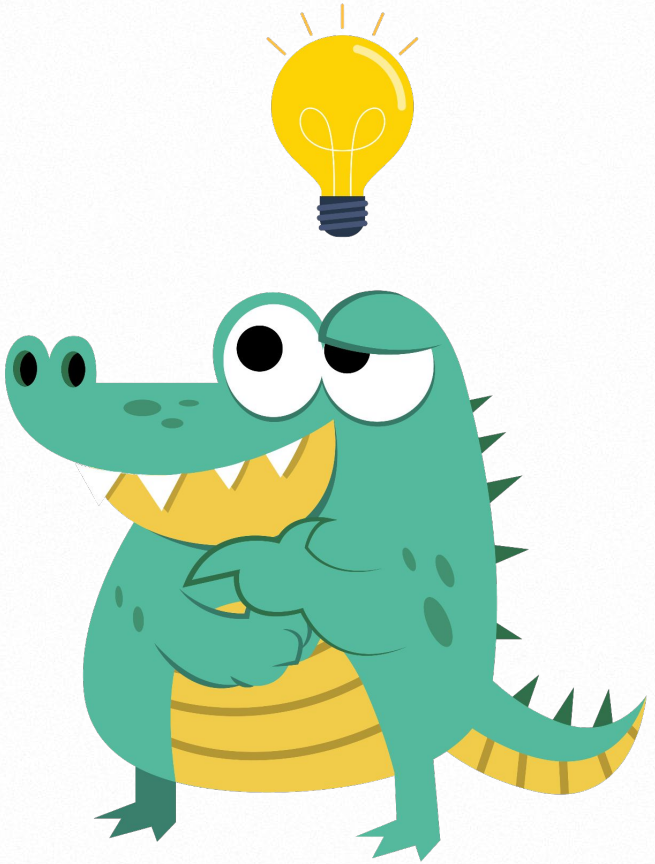
- **Esto abre la puerta a usar las funciones de flecha cuando no queremos que el contexto en el cuerpo de la función, cambie**

Paso por valor y paso por referencia

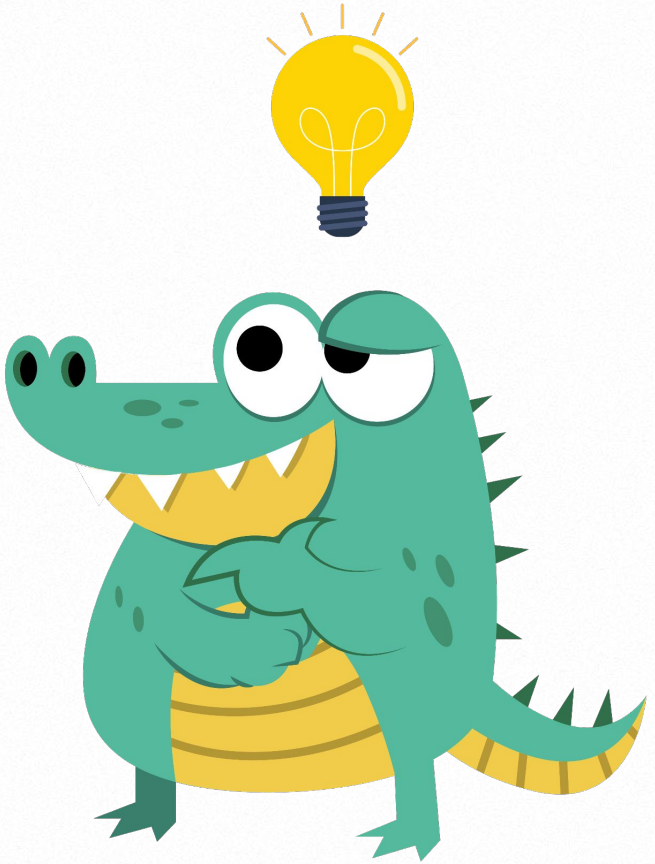
En JavaScript, un argumento puede ser enviado por valor o por referencia.



Decimos que un valor ha sido enviado como referencia, cuando el parámetro apunta a la misma dirección que el argumento original.



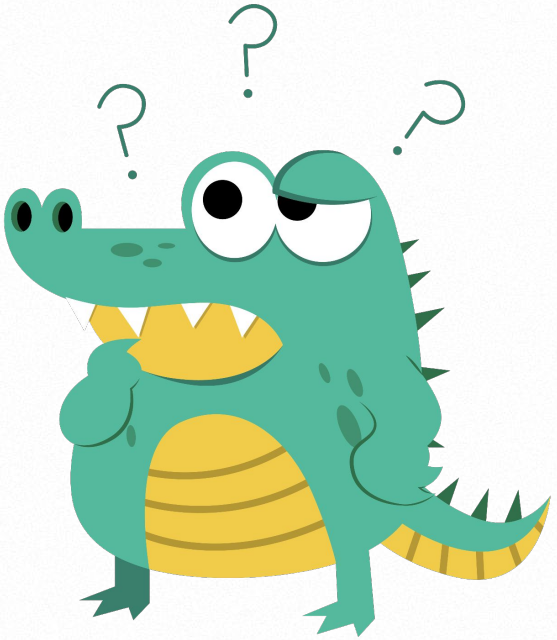
Decimos que pasamos por valor cuando el argumento enviado es copiado en una dirección distinta para el parámetro



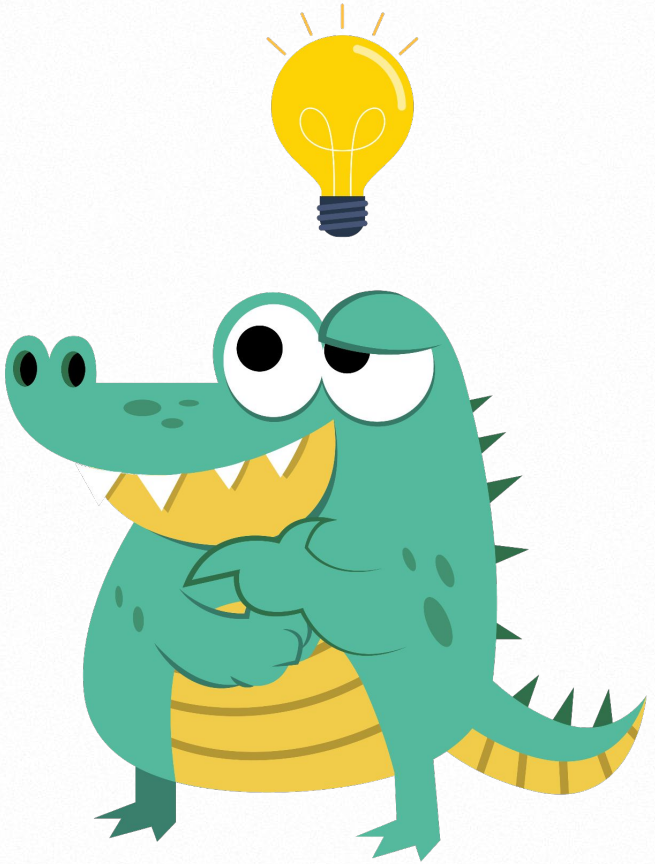
Veámoslo en práctica:

- Paso por valor y paso por referencia

¿Por qué esto es importante?

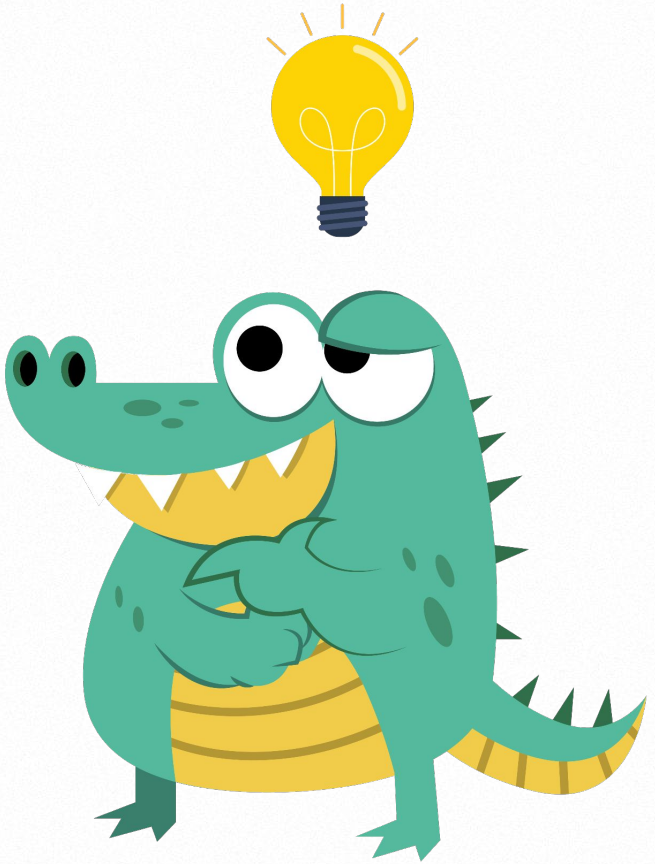


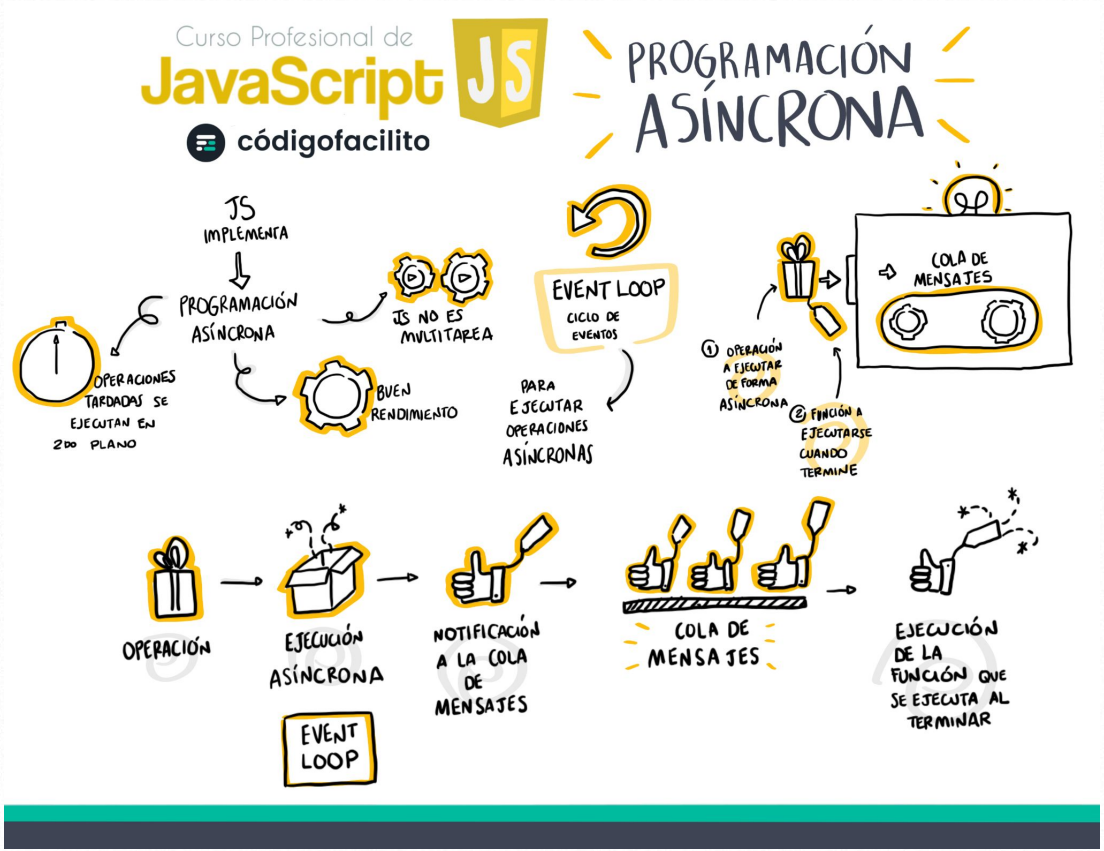
Durante tu trabajo con React aprenderás del concepto de “efectos secundarios”, y los valores pasados por referencia son una puerta a que una función produzca efectos secundarios.



Promesas

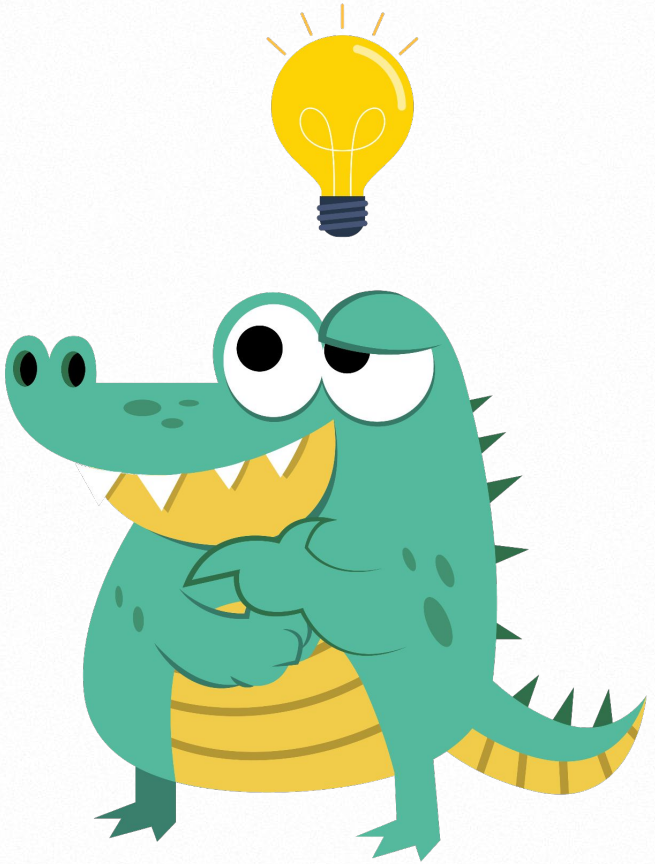
JavaScript es un lenguaje de programación asíncrono que se ejecuta sobre un solo hilo



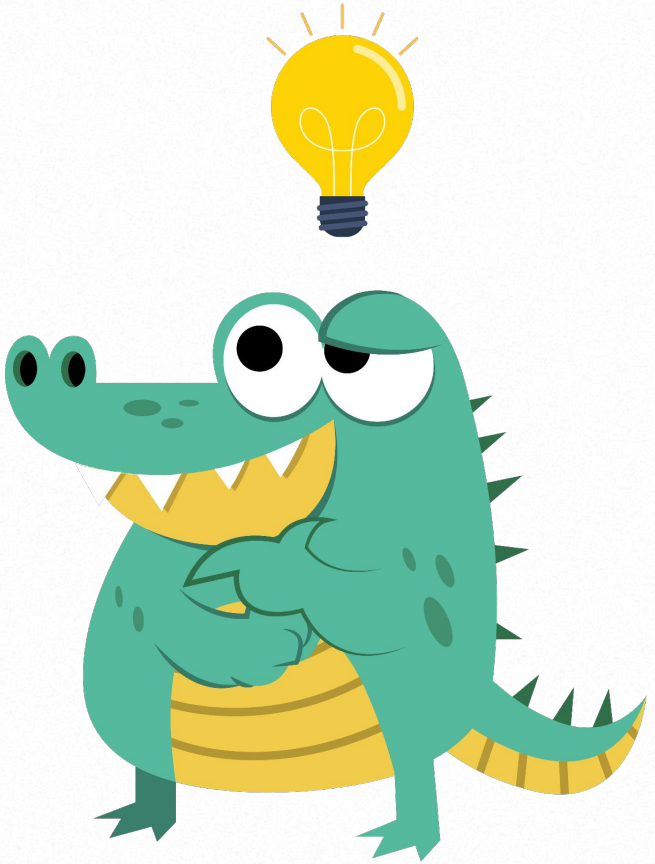


Para esto JavaScript introduce el event loop

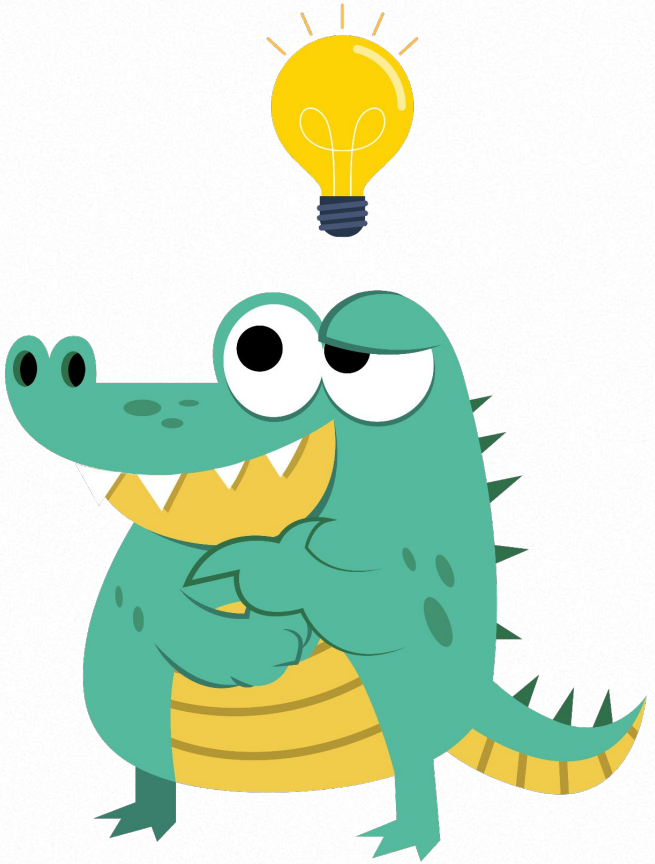
Para poder trabajar con el event loop JavaScript introduce el concepto de promesas, que nos permiten definir código a ejecutarse cuando una tarea fue finalizada.



Una promesa es un tipo de objeto retornado por una operación asíncrona, utilizando este objeto podemos obtener el valor final de la operación asíncrona, o sus errores.



Una promesa es un objeto que produce un valor en el futuro.



Una promesa puede encontrarse en alguno de los siguientes estados:

- fullfiled: O Completada, significa que la promesa se completó con éxito

- rejected: O rechazada, significa que la promesa no se completó con éxito

- pending: O pendiente que es el estado de la promesa cuando la operación no ha terminado, aquí decimos que la promesa no se ha cumplido.

- settled: O finalizada, cuando la promesa terminó ya sea con éxito o con algún error.

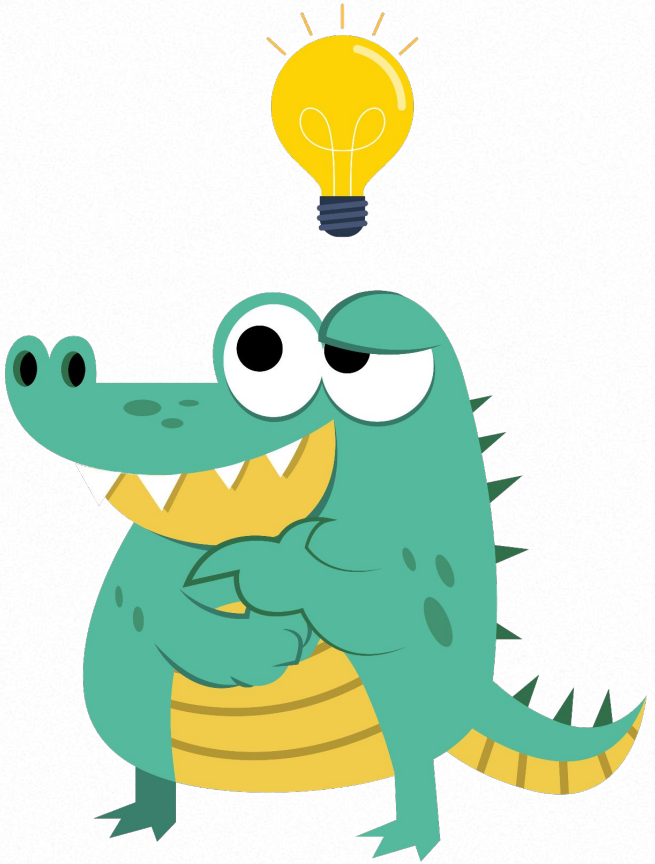
Veámoslo en práctica:

- Promesas
- Encadenar promesas

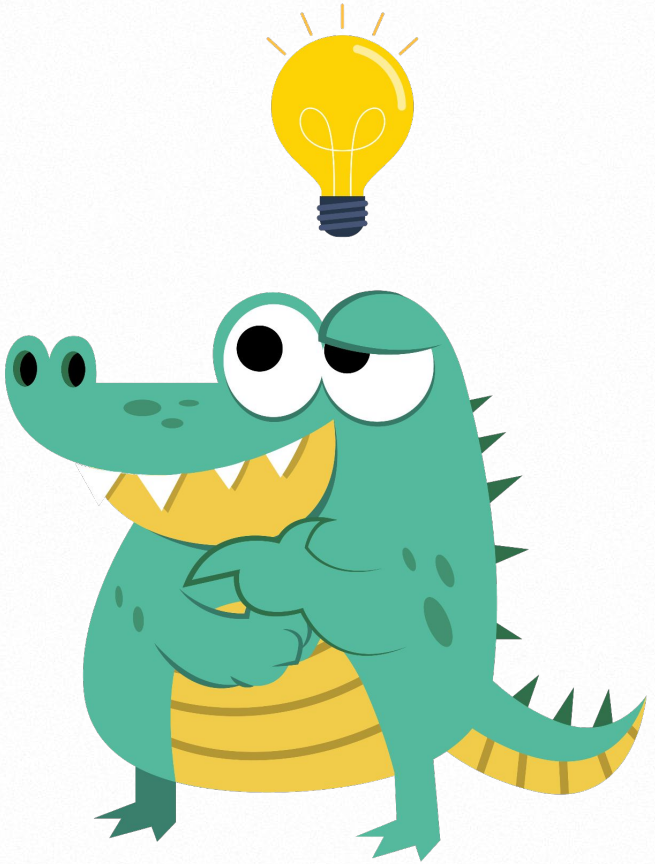
Promesas

Async/Await

Las funciones asíncronas permiten que usemos expresiones “await” dentro del cuerpo de la función.



Estas expresiones nos permiten trabajar con promesas como si fueran código síncrono.



Veámoslo en práctica:

- Async/Await