Introduccion a JavaScript moderno

¿Qué aprenderemos en esta sección?

Documentacion general: https://developer.mozilla.org/es/

- Generar la base sobre JavaScript
- Constantes y variables Let
- Template String
- Objetos literales
- Arreglos
- Desestructuración * (sumamente importante)
- Promesas
- Fetch API
- Ternarios
- Async - Await

Mi objetivo aquí es que tengamos las bases que nos ayuden a que podamos diferenciar fácilmente qué es propio de React y qué es propio de JavaScript. Estos conceptos y ejercicios nos ayudarán a suavizar la curva de aprendizaje de React.

Para trabajar estos conceptos vamos a montar una aplicacion de React.

1. Inicio de proyecto - Bases de JavaScript

Para iniciar con nuestra app de React debemos tener instalado nodeJS y npm (Node Package Manager).

Seguir las instrucciones de la siguiente pagina: https://create-react-app.dev/

2. Variables y constantes

Vamos a utilizar constantes para almacenar en memoria datos que no van a cambiar en el transcurso de la ejecucion.

let para variables que si pueden llegar a cambiar. Son variables de scope

var no se utiliza ya que es una mala practica y puede conllegar un monton de problemas su implementacion en proyectos a gran escala.

3. Template Strings

Los template strings permiten concatenar variables dentro de un string de manera mas facil. Se utiliza el operador `` y para colocar una variables dentro del mismo utilizamos la sintaxis ${VARIABLE}.

También reconoce los saltos de linea, operaciones de JavaScript, resultados de funciones, etc.

4. Objetos literales

Los objetos literales tambien son conocidos como diccionarios en otros lenguajes. En general estos objetos funcionan de manera clave:valor.

Los objetos literales son aquellos que en consola aparecen con las llaves {}. Los objetos comparten todos un Prototype, que son una serie de parametros, propiedades, metodos, etc.

Para copiar los valores de un objeto entre otros tenemos que recordar que hacer una asignacion = va a apuntar la variable nueva a la direccion de memoria del objeto a copiar, lo que afecte a una variable va a afectar a la otra. Para evitar este invonveniente utilizaremos el Spread Operator {...Objeto} que va a copiar los valores dentro del objeto.

5. Arreglos

Documentacion map:
https://developer.mozilla.org/es/docs/web/javascript/reference/global_objects/array/map

Un arreglo es una coleccion de informacion que se encuentra dentro de una misma variable. Es dinamica y no generica. Los array utilizan las llaves [] para identificarse en la consola.

array = new Array(CANTIDAD)

array = new Array() -> Da un warning

array = [] -> Soluciona el warning

6. Funciones

Las funciones declaradas explicitamente pueden ser sobre escritas con valores nuevos.

Por ejemplo:

```
function Saludar(nombre) {return `Hola, ${nombre}`};

saludar = 30;
```

Puede generar conflictos, va a mostrar 30 si hacemos un log de la variable saludar pero nos va a advertir que es una funcion.

Con las funciones declaradas podemos evitar este tipo de sobreescrituras y en vez de un warning nos va a romper el programa con un error.

7. Desestructuración de objetos/arreglos o Asignación destructurante

Documentacion:
https://developer.mozilla.org/es/docs/web/javascript/reference/operators/destructuring_assignment

La sintaxis de desestructuración es una expresión de JavaScript que permite desempacar valores de arreglos o propiedades de objetos en distintas variables.

8. Import, export y funciones comunes de arreglos

Documentacion:
https://developer.mozilla.org/es/docs/web/javascript/reference/global_objects/array/find

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/export

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/import

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Para importar archivos JavaScript a mi archivo actual utilizamos el comando import {NOMBRE_EXPORT} form 'PATH_RELATIVO';

Para exportar un archivo este tiene que tener export en cada modulo a exportar. Lo mismo si importamos tiene que ser con el nombre de cada export.

Si importamos un archivo por defecto no hace falta especificar el mismo nombre del export, podemos ponerle el nombre que querramos.

9. Multiples exportaciones y exportaciones por defeto

Distintas maneras:

```
export const  owners = ['DC', 'Marvel'];

export default heroes;

export {
   heroes as default,
   owner
}
```

10. Promesas

Documentacion: https://developer.mozilla.org/es/docs/web/javascript/reference/global_objects/promise

Una promesa es un objeto que dentro tiene dos callbacks. Es un objeto que representa la terminación o el fracaso de una operación asíncrona.

Mientras la promesa no haya terminado de ejecutar el codigo se encuentra en estado de pendiente. Una vez ejecutado el codigo si se puede resolver ejecutamos el callback resolve, resolve(OBJETO A PASARLE AL THEN); utilizamos la funcion .then() para procesar la información del resolve, en caso de que no se haya podido procesar ejecutamos el callback rejected, rejected(OBJETO A PASARLE AL CATH); utilizaremos la funcion .catch() para procesar la informacion del rejected. Existe el .finally() que va a ejecutar el codigo al final de la promesa sea cual sea su estado.

11. Fetch API

https://developer.mozilla.org/es/docs/Web/API/Fetch_API

La API Fetch proporciona una interfaz para recuperar recursos (incluso a través de la red).

Esta basado en Promesas, por lo cual tiene un response y un reject internos.

12. Async-Await

async crea una función asincrona, esta funcion asincrona no va a bloquear el event loop/stack y cuando termine se va a mostrar el codigo de la funcion async. Si tenemos que recibir informacion que puede llegar a tardar varios segundos es importante aplicar este concepto para que la página no se cuelge.

Si una función depende de otra asincrona ésta función debera ser asincrona también (ya que al depender de otra asincronia no se ejecuta en secuencia) y al llamar a la función asincrona original ponemos un await delante de la función para indicarle que espere a que termine el proceso y después continue con el código restante.

13. Operador de asignación ternario

Sintaxis: CONDICION ? TRUE : FALSE