Motivaciones

> Porque

Indice

Javascript orientado a React

Declarando variables

volver al indice

Var

Era el unico modo para declarar una variable, pero presentaba varios inconvenientes. Muchos programas antiguos la utilizan pero no es lo recomendable.

```
var foo = "bar";
/* Las variables declaradas con 'var' tienen un scope global
 a menos que fuese declarada dentro de una funcion */
```

Const

Es la forma en que declaramos variables constantes en javascript. Datos sobre `const`:

- Tiene un alcance de bloque, quiere decir que solo existe en el bloque que fue creada

```
const funcionFoo = () => {
  const foo = "bar";
  return foo;
};
funcionFoo();

console.log(foo); /*Mostraria un error de referencias pues esta fuera de su scope,
si quisieramos usar ese valor deberiamos asignar el resultado de la funcion a otra
variable. /*
```

- Solo pueden declararse una vez *(Dentro de su mismo scope)*

```
```

```
const foo = 1;
const foo = 3;
//Muestra un error que te dice que 'foo' ya ha sido declarado
```

- No puede cambiarse su valor y por lo tanto necesitas inicializarla

```
const foo = 1;
foo = "bar";
//Muestra un error de asignación
```

Los arrays y objetos declarados con const pueden ser modificados de ciertas formas

Let

> Variables ¿variables?

Al igual que **Const**, **let** se utiliza en las aplicaciones modernas para declarar variables y tiene alcance de bloque. Datos sobre `let`:

- Solo existe en su contexto:

```
if (esVerdad) {
  let foo = "bar";
}
console.log(foo); //undefined
```

- Solo puede declararse una vez en un mismo scope

```
let foo = "bar";
let foo = 2;
//Identifier 'foo' has already been declared.
```

- Puede variar su valor y por lo tanto no necesita inicializarse

```
let foo;
foo = "bar";

console.log(foo); //'bar'
```

TIP

> Puedes declarar varias variables usando una sola vez let *No lo recomiendo si no se hace de la forma adecuada* -> Ver buenas practicas

```
//Sin inicializar
let foo, bar, stuart;
//Inicializando
let foo = "hello",
  bar = "world",
  stuart = "little";
```

Exports

volver al indice

Con **export** podemos exportar funciones, objetos o tipos de datos para que despues puedan ser importados con imports

Básicamente existen dos tipos de exportaciones:

- Export sencillo

```
export const foo = "bar";

export let foo = "bar";

export class Clase {}

//Usando algo declarado previamente

const pin = 1234;

export { pin };
```

Este tipo de export es obligatorio importarlo con el nombre correspondiente. *Se le puede agregar un alias*

- Export por default

```
export default function(){...}
```

```
export default class Clase(){ }

const foo = 'bar';

export default foo;
```

Las variables no pueden usar `export default` directamente

Solo puede existir un export por defecto por archivo

TIP

> Si uno quiere usar una estructura de archivos basado en indices es interesante conocer que se puede re-exportar desde otro archivo.

```
//Re-exportamos bar.
export { bar } from "./bar.js";
//Esto seria equivalente
import { bar } from "./bar.js";
export { bar };
```

bar fue re-exportado de forma sencilla y deben ser importados con ese nombre y con el tipo de `import` adecuado.

Imports

volver al indice

Con **import** podemos importar funciones, objetos o tipos de datos que fueron efectivamente exportados con export

- Importando export por defecto de un modulo

```
import foo from "./bar.js";
//Le asignamos un nombre a esa importación, no necesariamente el mismo que tenia en
el archivo
import superiorFoo from "./bar.js";
//Si queremos importar algo más ademas de lo exportado por defecto
import foo, { bar } from "./bar.js";
```

- Importando los export sencillos

```
import { foo } from "bar.js";
```

```
```

Anatomia de estos import:

- `import` *Declaración de importación estática*
- `{foo}` *Modulo a importar, debe ir entre llaves y ademas haber sido exportado con ese nombre en especifico, ej:* `export {foo}`
- `from` *Desde*
- `'bar.js'` ruta al archivo

Funciones

volver al indice

Function declaration

Se *declara* una funcion con la palabra reservada `function`

```
function multiply(a, b) {
  return a * b;
}
```

Function expression

Se almacena la funcion en una variable, su principal ventaja y desventaja a la vez es que no puede ser usada hasta ser declarada, por lo tanto evitamos comportamientos inesperados.

```
multiply(2, 3); // No definida
const multiply = function (a, b) {
  return a * b;
};
multiply(2, 3); // 6
```

Arrow function

Las `arrow functions` siempre son anonimas, tambien se pueden asignar a una variable o usarse como callback. `() => {}`

```
const multiply = (a, b) => {
  return a * b;
};
```

```
multiply(2, 3); //6

/* El retorno en las arrow function puede ser implicito
siempre que se vaya a retornar algo inmediatamente
y no se vaya a agregar otra logica */

const multiply = (a, b) => a * b;

multiply(2, 3); // 6

/* Si solo recibe un parametro se pueden omitir
los parentesis */

const upName = (name) => name.toUpperCase();
upName("Fulano"); // 'FULANO'

/* Y son muy utiles para usar como callback */

otraFuncion(() => console.log('Hola'));

//otraFuncion solo espera un callback como parametro

/* Si solo se quiere retornar un objeto, este
se envuelve con parentesis */

const objectG = () => ({ title: "El hijo del consul", precio: 300 });

objectG(); //{title: 'El hijo del consul', precio: 300}
```

;

Timers

Convenciones

Asincronismo

Desestructuración

Objetos

Arrays

Buenas practicas

Programación funcional

Condicionales

Breve introduccion a node.js

Instalación

nvm

npm

pnpm

Github

volver al indice

`git init`

`git add`

`git clone`

`git push`

`git remote`

`git checkout`

`git branch`

Sass

Uso con CRA

Convenciones

Metodologias

Funciones y Mixins

Complementando con css-vars

---

volver al indice

React

volver al indice

Iniciar un proyecto

Convenciones

Uso de librerias externas

Bootstrap

Material Ui

Classnames

Craco

React-icons

Componentes

React router dom

Props

Ciclo de vida

---

useContext

---

useState

---

useEffect

---

useReducer

---

useRef

---

useMemo

Eventos

---

useCallback

---

useLayoutEffect

---

useImperativeHandle

Custom hooks

Proptypes

Deploy

---

vercel

heroku

netlify

Styleguidist

Patrones avanzados

React.lazy

Testing con react-testing-library

Redux

Virtual Dom

Typescript en react

volver al indice

Tipos

Interfaces

Genericos

Ventajas

VSCode 2022

volver al indice

Plugins

Temas

Configs