REACT.js

CLASE 2

Métodos de arrays

forEach

Ejecuta la función callback una vez por cada elemento presente en el array en orden ascendente.

```
unArray.forEach(function(value, index){
  console.log("En el indice: " + index + " está el valor: " + value);
});
```

find

Devuelve el **valor** del primer elemento del array que cumple la función de prueba proporcionada. En caso de no encontrarla, devuelve undefined.

```
var inventario = [
    {nombre: 'manzanas', cantidad: 2},
    {nombre: 'bananas', cantidad: 0},
    {nombre: 'cerezas', cantidad: 5}
];
```

```
inventario.find(function(fruta){
  return fruta.nombre == 'cerezas'
});
```

```
// { nombre: 'cerezas', cantidad: 5 }
```

map

map llama a la función callback provista **una vez por elemento** de un array, en orden, y construye un nuevo array con los resultados.

map

```
var results = [10,8,9,31].map(function(value){
  return value % 2;
});
```

```
// [0,0,1,1]
```

map

¿Y si quisiera los resultados por separado?

```
var results = [10,8,9,31].map(function(value){
  return value % 2;
});

first = results[0];

second = results[1];

third = results[2];
```

map

¿Y si quisiera los resultados por separado?

Destructuring

ES6

```
var results = [10,8,9,31].map(function(value){
  return value % 2;
});

[first, second, third] = results;
```

map

¿Y si quisiera los resultados por separado?

Destructuring

ES6

```
var results = [10,8,9,31].map(function(value){
  return value % 2;
});

[first, second, ...rest] = results;
```

map

¿Y si quisiera los resultados por separado?

Destructuring - **También funciona con objetos!**

ES6

```
var o = {p: 42, q: true};
var {p, q} = o;
```

filter

filter llama a la función dada callback para cada elemento del array , y construye un nuevo array con todos los valores para los cuales callback retorna un valor verdadero.

```
["🐮", "📣","🐔", "🌽"].map(isVegetarian)

//["🍟","🍿"]
```

filter

```
[10,8,9,31].filter(function(value){
  return value%2==0;
});
```

```
// [10,8]
```

reduce

El método **reduce()** aplica una función a un acumulador y a cada valor de un array (de izquierda a derecha) para reducirlo a un único valor.

```
["🍔", "🍟","🍗", "🍿"].reduce(eat)

//["💩"]
```

reduce

```
[3,4,10].reduce(function(accumulator,currentValue){
  return accumulator + currentValue;
},[initialValue]);
```

```
// 17
```


Ejercicio 1
Tiempo de práctica

Funciones

Definición de una función

Declaramos una función con un nombre para hacer referencia e invocar la función.

```
function cuadrado(lado) {
    return lado*lado
}
```

```
cuadrado(5)
//25
```

Expresión de una función

Asignamos una función anónima a una variable. Podemos invocar la función utilizando la variable.

```
var cuadrado = function(lado) {
    return lado*lado
}
```

```
cuadrado(5)
//25
```

function declarations vs function expressions

```
anterior(5);
doble(5);

var doble = function(num){
    return num*2;
};

function anterior(num){
    return num-1;
}
```

function declarations vs *function expressions*

```
anterior(5);      //4
doble(5);         //Error

var doble = function(num){
    return num*2;
};

function anterior(num){
    return num-1;
}
```

Scope

El alcance de una variable determina su accesibilidad.

```
function next(){
    var a = 6;
    return a+1;
}
```

“a” es variable LOCAL.

```
var a = 6;
function next(){
    return a+1;
}
```

“a” es variable GLOBAL.

Closure

```
function padre(){

   var a = 1;

   function closure(){
       console.log(a);
   }

   closure();
}
```

Cuando al anidar funciones, una función crea una variable local y una función interna, ésta función interna es un closure y solo está disponible dentro de la función padre.
A diferencia de la función padre, el closure no tiene variables locales y usa las declaradas dentro de padre().

Closures

```
function precioFinal(precioNeto){

   function impuestoIVA(){
      return precioNeto * 0.21;
   }

    return precioNeto + impuestoIVA();
}
```

Closures

```
function saludoConVar() {
    var saludo = "Hola";
    if (true) {
        var saludo = “Chau”;
        console.log(saludo);
    }
    console.log(saludo);
}
```

¿Qué imprime por consola?

Closures

```
function saludoConVar() {
    var saludo = "Hola";
    if (true) {
        var saludo = "Chau";
        console.log(saludo);   //Chau
    }
    console.log(saludo);       //Chau
}
```

¿Qué imprime por consola?

Let ES6

Permite declarar variables limitando su alcance (scope) al bloque, declaración o expresión donde se está usando.

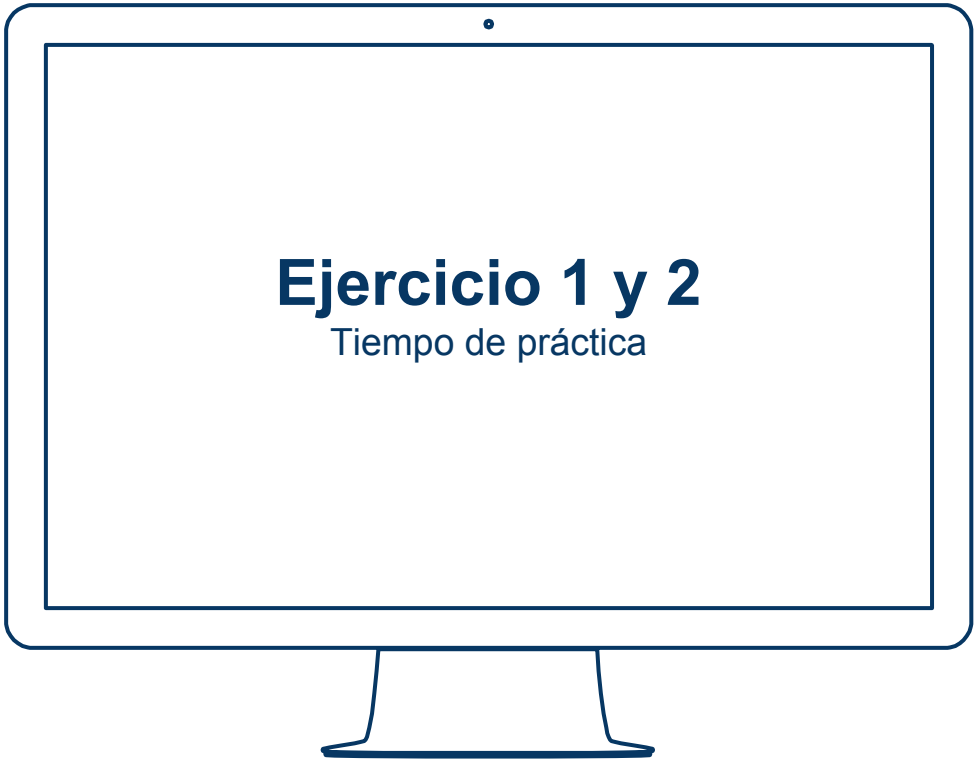
Let

¿Qué diferencia hay con var?

ES6

```
function saludoConLet() {
    let saludo = "Hola";
    if (true) {
        let saludo = "Chau";
        console.log(saludo);   //Chau
    }
    console.log(saludo);       //Hola
}
```

¿Qué imprime por consola?


Ejercicio 1 y 2
Tiempo de práctica

Podemos tratar a las funciones como variables.
Esto nos da la posibilidad de:

```
var doble = function(a){
      return a*2;
}
```

Almacenarla en variable

```
function vuelto(monto){
   precio = 100;
   function calcVuelto(){
     return precio-monto;
   }
   return calcVuelto();
}
```

Retornarla

```
function a(){
   console.log("a");
}
function b(a){
   console.log("b");
   a();
}
```

Pasarla como argumento

¿Qué hace esto?

```
function username(unaFuncion){
     var name = prompt("Ingrese su nombre");
     unaFuncion(name);
}

function welcome(name){
     alert("Bienvenido " + name + "!");
}
```

username(welcome);

Callback

Callback es una función que se pasa por parámetro a otra función y en principio se ejecuta una vez que se haya terminado de ejecutar la función anterior.

Cuando enviamos una función anónima a un evento, estamos utilizando Callbacks.

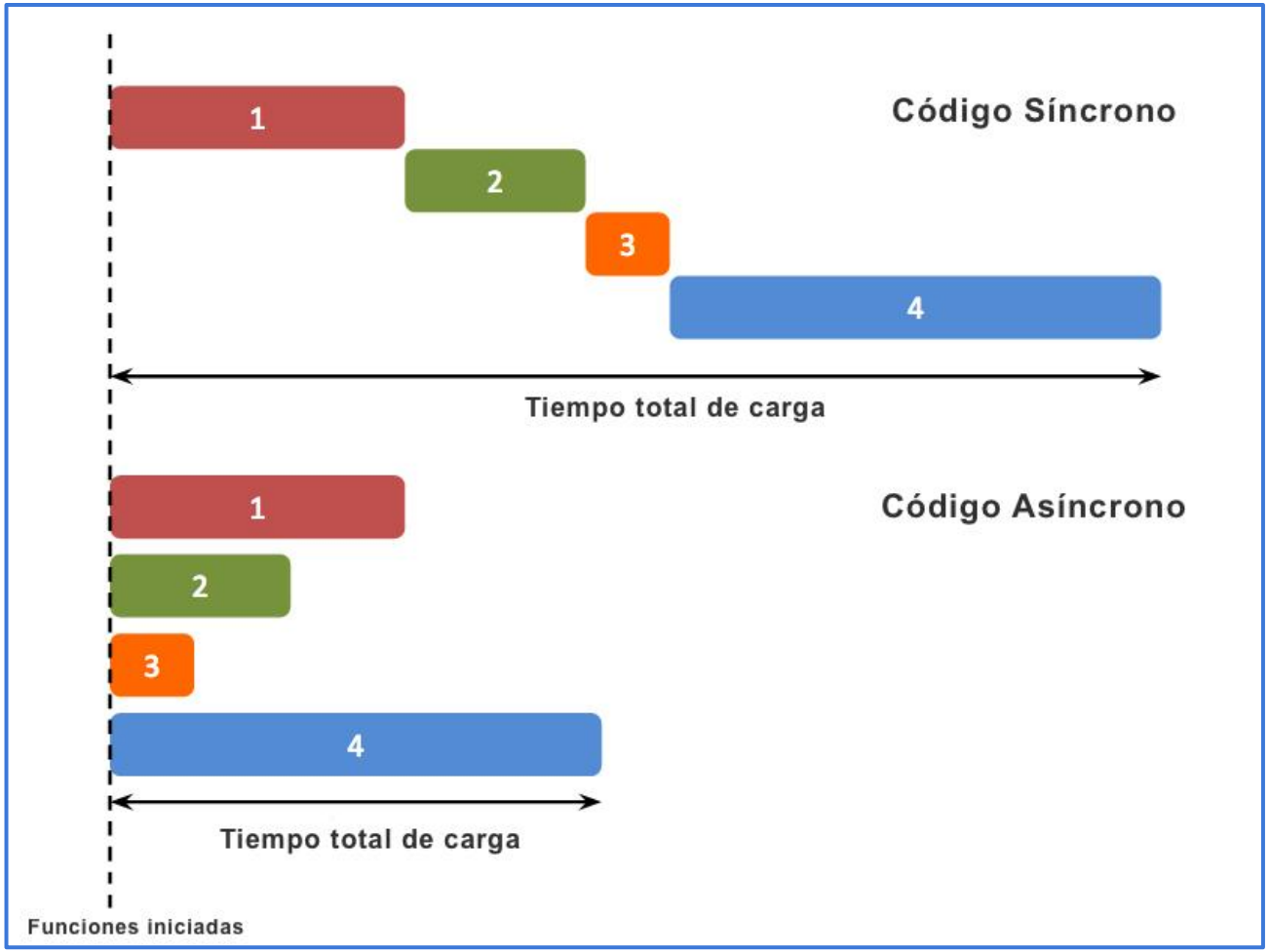
Un Callback funciona como un Closure.

¿Qué hace esto?

```
function funcionA(funcionB){
     console.log("a");
     setTimeout(function(){
          funcionB()
     },1000);
     console.log("c");
}

function funcionB(){
     console.log("b");
}

funcionA(funcionB);
```


Código Síncrono
1
2
3
4
Tiempo total de carga
Código Asíncrono
1
2
3
4
Tiempo total de carga
Funciones iniciadas

Asincronía en JS

Asincronía en JS

- Callbacks (ES5)
- Promises (ES6)
- Async/await (ES7)

Callbacks

Callback -> Procesos asincrónicos

```
function primero(callback){
     setTimeout(function(){
          callback();
     },1000);
}

function segundo(){
     console.log("Ejecutando callback");
}
```

```
console.log('antes');
primero(segundo);
console.log('despues');

//antes
//despues
//Ejecutando callback
```

Callback -> Procesos asincrónicos

```
function primero(callback){
     setTimeout(function(){
          console.log("Primero");
          callback();
     },1000);
}

function segundo(callback){
     setTimeout(function(){
          console.log("Segundo");
          callback();
     },1000);
}

function tercero(){
     console.log("Tercero");
}
```

primero(segundo(tercero));

Callback hell

```
primero(function(){
     segundo(function(){
          tercero(function(){
               cuarto(function(){
                    ...
               });
          });
     });
});
```


Ejercicios
Tiempo de práctica

Promises

Promises

Una Promesa es un objeto que representa la terminación o el fracaso eventual de una operación asíncrona.

Representa un valor que puede estar disponible ahora, en el futuro o nunca.

Promises

```
const promise = f();

promise.then(exitoCallback).catch(errorCallback);
```

Promises

```
function f(){
    return (new Promise((resolve, reject) => {
        var ok = Math.round(Math.random());
        if(ok){
            var foo = "Todo ok";
            resolve(foo);
        } else {
            reject(new Error("Ocurrio un error"));
        }
    })
)}
```

Promises

```
function f(){
    return (new Promise((resolve, reject) => {
        var ok = Math.round(Math.random());
        if(ok){
            var foo = "Todo ok";
            resolve(foo);
        } else {
            reject(new Error("Ocurrio un error"));
        }
    })
)}
```

```
const promise = f();
promise.then(foo => console.log(foo)).catch(error => console.log(error));
```

Promise chaining

```
const promise = f();

promise
    .then(function(response1){ //código })
    .then(function(response2){ //código })
    .then(function(response3){ //código })
    .then(function(responseN){ //código })
    .catch(errorCallback);
```

Const

ES6

Puede recibir un valor en el momento de la declaración.
Luego no puede ser modificada.

```
const pi = 3.14;
pi = 3.141592635;   //Error!!
```


Ejercicios
Tiempo de práctica

AXIOS

AXIOS

AXIOS es una librería de Javascript para hacer pedidos asincrónicos HTTP (AJAX) a través de promesas

AXIOS

```
npm install axios
```

```
import Axios from 'axios';
```

AXIOS - GET

```
Axios.get(url)
    .then(callbackSuccess)
    .catch(callbackFail);
```

Ejemplo:

```
Axios.get('https://pokeapi.co/api/v2/pokemon')
    .then((response)=>console.log(response))
    .catch((error)=>console.log(error));
```

AXIOS - POST

```
Axios.post(url, data)
    .then(callbackSuccess)
    .catch(callbackFail);
```

Ejemplo:

```
Axios.post('/user', {name:'Pepe', age:'18'})
    .then((response)=>console.log(response))
    .catch((error)=>console.log(error));
```


Ejercicios
Tiempo de práctica

Gracias!

¿Preguntas?