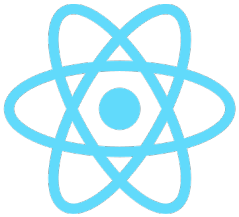
# Desarrollo y programación de Aplicaciones con React

## 1. ECMAScript

core networks



# Desestructuring

Es una funcionalidad de JS a partir de ECMAScript y sucesivas que permite desestructurar arrays y objetos a partir de sus elementos o propiedades.

Sintaxis básica para arrays

const [identificador1, identificador2, ...resto] = [elemento1, elemento2, ...resto elementos];

# Desestructuring

Es una funcionalidad de JS a partir de ECMAScript y sucesivas que permite desestructurar arrays y objetos a partir de sus elementos o propiedades.

Sintaxis básica para objetos

const {identificador1, identificador2, ...resto} = { propiedad1: valor, propiedad2: valor, ...}

# Arrow functions

Una nueva sintaxis para funciones de expresión o anónimas

Sintaxis básica

```
(parámetros) => {
    código del cuerpo de la función
}
```

# Arrow functions

Dispone de simplificaciones para los siguientes casos

- Si solo tiene un parámetro se pueden aliviar los paréntesis
- Si solo tiene una línea de código en el cuerpo de la función se pueden aliviar las llaves
- Si solo tiene una línea de código return se puede aliviar por la evaluación de la expresión
- // Si no tuviera parámetros los paréntesis se pueden

cambiar por underscore _

Otra de sus características es que no dispone de this.

# Funciones de alto nivel para arrays mas usadas

array.forEach(callback)

array.map(callback)

array.filter(callback)

# Asincronía en JavaScript I. Funciones callback.

La primera forma de manejar asincronía en JS es mediante callbacks que se fundamentan en el uso de funciones como argumentos de otra función, lo que permite invocarlas dentro de la función externa para llevar a cabo algún tipo de acción.

```
function funcionExterna (parámetros, callback) {
        // otro código;
        callback(parámetros); // Se invoca dentro de la función externa
}
```

# Asincronía en JavaScript II. Promesas.

La otra forma de manejar asincronía en JS es mediante promesas. Una promesa es esencia un objeto que en su instancia usa una función con dos parámetros, resolve y reject, que son usados para completar o rechazar una tarea asíncrona.

```
new Promise(function(resolve, reject) {
        // código con
        reject();
        // código con
        resolve();
})
```

# Asincronía en JavaScript II. Promesas.

La ventaja fundamental de las promesas es su consumo, que implementa los métodos then() para recibir el valor de resolve() y catch() para recibir reject().

```
funcionRetornaPromesa()
        .then()
        .then() // Los then se pueden encadenar
        .catch()
```