JavaScript

Facundo Goyena - 2017

JavaScript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos,3 basado en prototipos, imperativo, débilmente tipado y dinámico.

Wikipedia.org

JavaScript - Scope

- Alcance que tiene una variable o función - Global o Local

```
var prueba = 1; // variable global

console.log(prueba); // 1

function funcionPrueba() {
    prueba = 2;
    var local = 5;
}

funcionPrueba();

console.log(prueba); // 2
console.log(local); // undefined
```

https://www.w3schools.com/js/js_scope.asp

JavaScript - Hoisting

- Las variables pueden ser utilizadas antes de ser declaradas

Funciona:

```
prueba = 1;

console.log(prueba); // 1

var prueba;
```

No Funciona:

```
console.log(prueba); // undefined

var prueba = 1;
```

https://www.w3schools.com/js/js_hoisting.asp

JavaScript - Data Types

```
var numero = 1;                    // number
var cadena = 'cadena de texto';    // string - definido con '
var cadena2 = "cadena de texto";   // string - definido con "
var personaJuanPerez = {           // object
    nombre: 'juan',
    apellido: 'perez'
};
var booleano = true;               // boolean
var indefinido;                    // undefined
var nulo = null;                   // null
function funcionPrueba() {}        // function


typeof numero;                     // number
typeof nulo;                       // null
typeof funcionPrueba;              // function
```

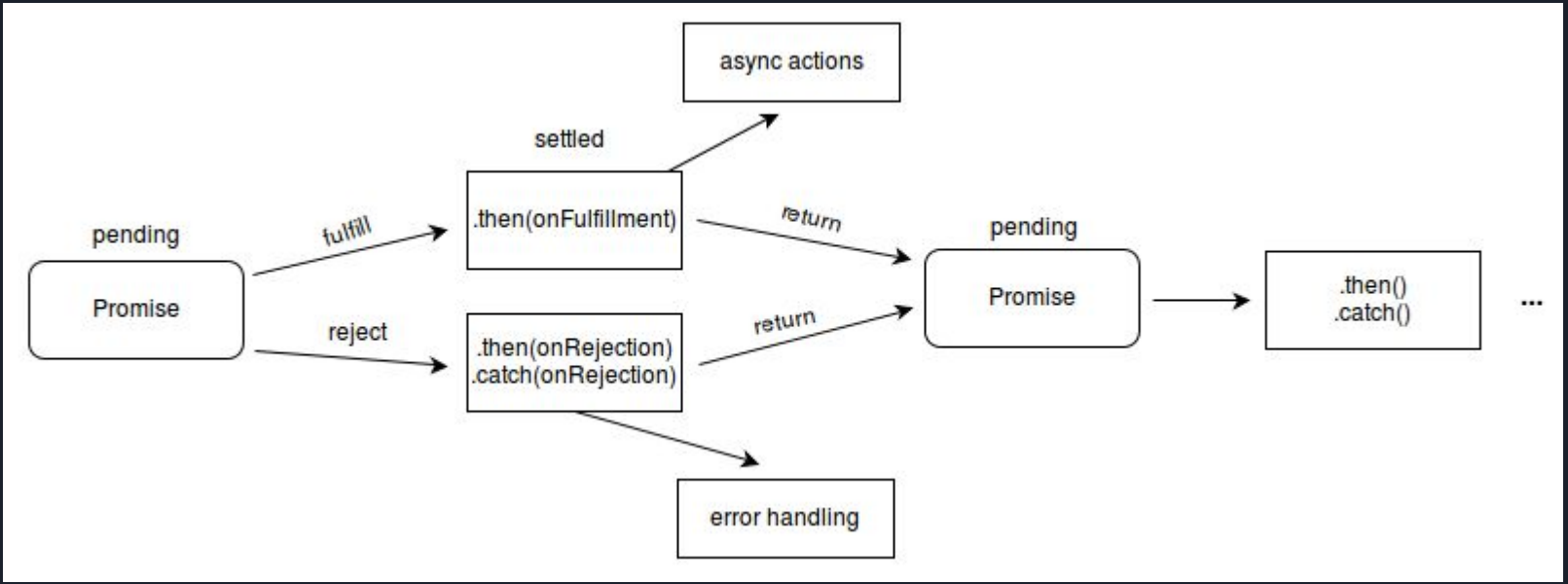
https://www.w3schools.com/js/js_datatypes.asp

JavaScript - Promesas

- Utilizado para operaciones asíncronas
- Posee diferentes estados
  - Pendiente
  - Cumplida
  - Rechazada
- Mediante *callbacks* se ejecutan las acciones correspondientes

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

JavaScript - Promesas



https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

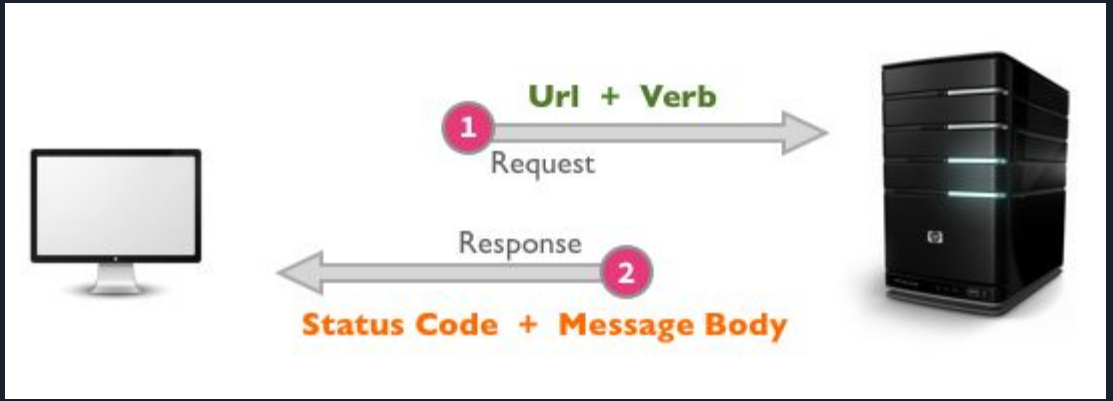
JavaScript - Promesas: Ejemplo

```
let miPrimeraPromise = new Promise(function (resolve, reject) {
  // Llamamos a resolve(...) cuando lo que estabamos haciendo finaliza con éxito, y reject(...)
cuando falla.
  // En este ejemplo, usamos setTimeout(...) para simular código asíncrono.
  // En la vida real, probablemente uses algo como XHR o una API HTML5.
  setTimeout(function(){
    resolve("¡Éxito!"); // ¡Todo salió bien!
  }, 250);
});

miPrimeraPromise.then(function (successMessage) {
  // succesMessage es lo que sea que pasamos en la función resolve(...) de arriba.
  // No tiene por qué ser un string, pero si solo es un mensaje de éxito, probablemente lo sea.
  console.log("¡Sí! " + successMessage);
});
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

JavaScript - Peticiones HTTP



JavaScript - Verbos HTTP

HTTP Methods and Their Meaning

Method	Meaning
GET	Read data
POST	Insert data
PUT or PATCH	Update data, or insert if a new id
DELETE	Delete data

JavaScript - XMLHttpRequest

```
var xhr = new XMLHttpRequest();
xhr.open("GET", "https://jsonplaceholder.typicode.com/users",
true);
xhr.onload = function (e) {
  if (xhr.readyState === 4) {
    if (xhr.status === 200) {
      console.log(xhr.responseText);
    } else {
      console.error(xhr.statusText);
    }
  }
};
xhr.onerror = function (e) {
  console.error(xhr.statusText);
};
xhr.send(null);
```

https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests

JavaScript - API de ejemplo

- https://jsonplaceholder.typicode.com/
  - https://jsonplaceholder.typicode.com/users

JavaScript - OOP

- Encapsulamiento
- Herencia
- Abstracción

Métodos de aplicación:

- Los objetos pueden funcionar como “clases estáticas” - tienen una única instancia
- Funciones “constructoras”

JavaScript - Objetos Literales

```
var juan = {
    nombre: 'Juan',
    apellido: 'Perez'
};

console.log(juan.nombre); // Juan
console.log(juan.apellido); // Perez
```

JavaScript - Funciones Constructoras

```
function Persona(nombre) {
    var _nombre = nombre;

    this.saludar = function () {
        alert('Hola, soy ' + _nombre);
    };
}

var juan = new Persona('Juan');

juan.saludar(); // alert('Hola, soy Juan')
```

```
function Persona(nombre) {
    this.nombre = nombre;
}

Persona.prototype.saludar = function () {
    alert('Hola, soy ' + this.nombre);
};

var juan = new Persona('juan');

console.log(juan.nombre); // juan
juan.saludar(); // alert('Hola, soy Juan')
```

JavaScript - Encapsulamiento

```
function Persona(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
}

Persona.prototype.cumple = function () {
    this.edad += 1;
};

var juan = new Persona('juan', 20);

console.log(juan.edad); // 20
juan.cumple();
console.log(juan.edad); // 21
```

JavaScript - Clases con ES6

```
class Persona {
    constructor(nombre) {
        this.nombre = nombre;
    }

    saludar() {
        alert('Hola, soy ' + this.nombre);
    }
}

let juan = new Persona('juan');

console.log(juan.nombre); // juan
juan.saludar(); // alert('Hola, soy juan')
```

```
class Perro {
    constructor(raza) {
        this.raza = raza;
    }

    ladrar() {
        return 'guau';
    }
}

class Beagle extends Perro {
    constructor(nombre) {
        super('Beagle');
        this.nombre = nombre;
    }

    ladrar() {
        return 'yo tambien digo ' + super();
    }
}

let quinn = new Beagle('Quinn');
console.log(quinn.ladrar());
```

JavaScript - Clases con Getters y Setters

```
class Persona {
    constructor(nombre) {
        this._nombre = nombre;
    }

    get nombre() {
        return this._nombre;
    }

    set nombre(nuevoNombre) {
        this._nombre = nuevoNombre + 'cito';
    }
}

let juan = new Persona('juan');

console.log(juan.nombre); // juan
juan.nombre = 'juan';
console.log(juan.nombre); // juancito
```