**ES6 Compiler y Transpiler**

Con el tiempo los interpretadores de JavaScript han evolucionado para adaptarse a las necesidades de la construcción de software actuales. Aunado a estas nuevas versiones de interpretadores, vienen los problemas relacionados con la compatibilidad de los códigos que no podrían funcionar en navegadores que no implementen estas nuevas versiones. A continuación analizaremos estos problemas y la forma de solucionarlos.

En JavaScript tenemos muchas versiones de interpretadores, la más comúnmente implementada en los navegadores es la versión de **EcmaScript 5.1**, que se liberó a mediados del 2011. A esta versión le siguió**EcmaScript 6**(ES6), la cual fue liberada en el 2015, haciéndose cada vez más popular entre los navegadores modernos, quienes cumplen con dicha especificación casi completamente. Sin embargo, si utilizas instrucciones del ES6 tienes el riesgo de que tu código no se ejecute en versiones viejas de los navegadores. Para evitar esto, tenemos **Babel**, que es un compilador **source-to-source**, también llamado **transpilador** que convierte código ES6 a código JavaScript compatible (ej.: ES5.1).

Esta conversión se puede hacer en un proceso de preparación del código, o se puede configurar para que el navegador lo haga en vivo al cargar tus programas en ES6. A continuación, veamos un ejemplo de conversión.

En ES6 podemos definir clases. A continuación un ejemplo:

class Empleado {

contructor(nombre = undefined) {

this.nombre = nombre

}

saludar() {

return `Hola soy ${this.nombre}`

}

}

emp = new Empleado("luis")

console.log(emp.saludar())

Lo que se traduce usando Babel en ES5.1 en lo siguiente:

"use strict";

function \_classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) { throw new TypeError("Cannot call a class as a function"); } }

var Empleado = function () {

function Empleado() {

\_classCallCheck(this, Empleado);

}

Empleado.prototype.contructor = function contructor() {

var nombre = arguments.length > 0 && arguments[0] !== undefined ? arguments[0] : undefined;

this.nombre = nombre; }; Empleado.prototype.saludar = function saludar() {

return "Hola soy " + this.nombre;

};

return Empleado;

}();

emp = new Empleado("luis")

console.log(emp.saludar());

Aquí puedes encontrar la herramienta Babel [https://babeljs.io/](https://babeljs.io/" \t "_blank). Pero así como Babel, existen otras implementaciones de conversores.

Además de convertir a partir de ES6, existen otros lenguajes que se inspiran en JavaScript pero que hacen, en su parecer, mejoras. Por ejemplo, **CoffeeScript** es un lenguaje que promete escribir código más limpio usando el concepto de sangrado (o indentación), que se emplea en Python. Un ejemplo de CoffeeScript se muestra a continuación:

class Empleado

constructor: (@name) ->

saludar: ->

"Hola soy #{@name}"

emp = new Empleado("luis")

Lo anterior da como resultado el siguiente código en ES5.1:

var Empleado, emp;

Empleado = (function() {

function Empleado(name) {

this.name = name;

}

Empleado.prototype.saludar = function() {

return "Hola soy " + this.name;

};

return Empleado;

})();

emp = new Empleado("luis");

Otro lenguaje que está tomando mucha fuerza es **TypeScript**, el cual es definido por Microsoft y es una implementación de ES6 con algunos elementos añadidos, entre los cuales tenemos el manejo de tipos de datos explícitos (fuertemente tipado), lo cual se hace atractivo para desarrollos más formales.

A continuación un ejemplo de TypeScript donde se utiliza el tipo string para definir el atributo nombre.

class Empleado {

nombre: string

contructor(nombre: string) {

this.nombre = nombre

}

saludar() {

return `Hola soy ${this.nombre}`

}

}

emp = new Empleado("luis")

console.log(emp.saludar())

Aunque hoy en día se utilizan las versiones ECMAScript estudiadas en esta lectura, a medida que pasa el tiempo se implementan nuevas versiones de EcmaScript con sus respectivos transpiladores para adaptarse a las necesidades de los desarrollos de software recientes.