**Universidad Tecnológica Centroamericana**

**UNITEC**

**Tarea:**

ECAMAScript6

**Experiencia de Usuario**

Sección 2168

**Docente:**

Ing. Elmer Padila

**Alumna:**

Crysthel Aparicio Bonilla

**Cuenta:**

31511318

**Campus Tegucigalpa**

8 de agosto del 2019



ECMAScript 6

* ¿Qué es?

Es la próxima versión de JavaScript, y tiene algunas nuevas características excelentes. Las características tienen diversos grados de complejidad y son útiles tanto en scripts simples como en aplicaciones complejas. Se podría considerar que es una auténtica revolución en la sintaxis de Javascript. Su buque insignia es, probablemente, una clara orientación a clases y herencia, pero la verdad es que hay muchas otras novedades interesantes, como el uso de módulos, los parámetros por defecto, las variables let y const, o la novedosa sintaxis de las funciones arrow, entre otros cambios.

* ¿Por qué su importancia en los últimos años?

En 1999 aparece la 3a versión del estándar ECMAScript, que se mantendría vigente hasta hace pocos años. Hubo pequeños intentos de escribir la versión 4, pero hasta 2011 no se aprobó y se estandarizó la versíon 5 (ES5) que es la que usamos hoy en día.

En junio de 2013 quedó parado el borrador de la versión 6, pero en diciembre de 2014 se aprobó al fin y se espera su estandarización a partir de Junio de 2015.

Desde el lanzamiento en junio de 1997 del estándar ECMAScript 1, han existido las versiones 2, 3 y 5, que es la más usada actualmente (la 4 se abandonó). En junio de 2015 se cerró y publicó la versión ECMAScript 6.

* ¿Cuáles son las nuevas características o funcionalidades?

La sexta edición agrega cambios significativos en la sintaxis para escribir aplicaciones complejas, incluyendo clases y módulos, definiéndolos sémanticamente en los mismos términos del modo estricto de la edición ECMAScript 5. También implementa:

* Arrow functions
* Clases
* Template strings
* Let y const
* Generadores
* Literales octales y binarias
* Maps y sets
* Promises
* ¿Cómo funcionan estas nuevas funcionalidades?

Arrow functions

var cuadrados = numeros.map(n => n \* n);

Clases

class Rectangulo {

constructor(base, altura) {

this.base = base;

this.altura = altura;

}

calcArea() {

return this.base \* this.altura;

}

}

var r = new Rectangulo(5, 10);

console.log(r.calcArea()); // 50

Template strings

var s1= 'esta es una template string';

// Pueden contener valores

var n = 5;

var s2 = `El valor de n es ${n}`;

// Pueden abarcar múltiples líneas

var s3 = `Esta es una cadena

escrita en dos líneas`;

Let y const

**Let** indica que una variable solo va a estar definida en un bloque en particular, al terminar el bloque la variable deja de existir, esto es muy útil para evitar errores lógicos cuando alteramos una variable que no deberíamos.

function letTest() {

if (true) {

let x = 23;

console.log(x); // 71

}

console.log(x); // no existe x

}

Const por su parte previene que una variable declarada cambie de valor, convirtiéndola efectivamente en una constante.

const a = 7;

a = 5; // error

console.log(a);

Generadores

Los generadores son un tipo especial de función que retornan una serie de valores con un algoritmo definido por el usuario, podríamos generar ids consecutivos para registros de una base de datos, sucesiones numéricas como [Fibonacci](https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci), entre otras.

function\* cuadrados(){

var n = 1; // comienza en 1

while(true) {

var c = n \* n; // obtiene el cuadrado

n++; // aumenta para la próxima iteración

yield c; // devuelve el valor actual

}

}

Literales octales y binarias

Hay ocasiones en que el contexto de nuestros datos requiere que trabajemos con cifras no decimales, por ejemplo en base 2 (binario) o base 8 (octal), ahora es sencillo crear este tipo de literales con los prefijos (0b)y (0o) respectivamente.

var a = 0b111110111; // binario

console.log(a); // 503

var b = 0o767; // octal

console.log(b); // 503

Maps y sets

Set puede encadenar el método add para añadir nuevos elementos, si alguno se duplica es omitido y tiene un método has para revisar si existe un elemento dentro del conjunto.

// Sets

var s = new Set();

// Añade 3 elementos, cadena1 se repite

s.add("cadena1").add("cadena2").add("cadena1");

// El tamaño es 2

console.log(s.size === 2);

// El conjunto no tiene la cadena hola

console.log(s.has("hola"));

Un mapa es similar, pero para añadir un elemento usamos el método set que acepta dos parámetros: la llave y su valor asociado. Se obtienen los valores con el método get.

// Maps

var m = new Map();

// Añade la llave "hola" con el valor 42

m.set("hola", 42);

// Añade la llave "a" con el valor 34

m.set("a", 34);

// Obtiene el valor asociado a la llave "a"

console.log(m.get("a")); // 34

Promises

Las promesas son objetos que representan esta clase de operaciones y los datos que se obtienen.

var promesa = obtenerDatos();

promesa.then(function (dato) {

console.log(dato); // 'mensaje'

}, function (error) {

console.error(error); // ocurrió un error

});