PROGRAMACIÓN AVANZADA EN JAVASCRIPT

Módulo 3

OTEC PLATAFORMA 5 CHILE

MÓDULO 3

CLASE 6

Introducción

setTimeOut

Callback

Promises

Async-Await

Introducción

En otras oportunidades hemos conversado sobre las funciones asincrónicas, las cuales nos permiten que continúe el proceso de ejecución sin importar si la función anterior no se haya completado.

Esta es una de las características que hace que JavaScript sea tan poderoso, ya que es un lenguaje que solo procesa en un hilo o pila.

Es decir, que no realiza múltiples procesos, sino ejecuta uno por vez.

En esta clase nos concentramos en dichas funciones.

Función setTimeOut

Le vamos a dejar el siguiente código para que lo analice y pruebe en su consola.

```
function sum(x, y) {
 console.log(x + y);
}

setTimeout(function () {
 sum(2, 2);
}, 1000);

sum(4, 4);
```

Se habrá dado cuenta que el número 8 fue mostrado primero que el 4. Esto se debe a que setTimeout es una función asíncrona que recibe una función como primer parámetro y la cantidad de milisegundos que será ejecutada dicha función.

¿Pero qué tal si lo cambiamos a 0 el segundo parámetro de setTimeout?

Nada, seguirá mostrando el 8 antes que el 4. Esto sucede porque no importa la cantidad de milisegundos que se le asignen, el setTimeOut siempre se va a ejecutar como última instancia.

Callback

Los callbacks son funciones que son pasadas como parámetro para ser ejecutadas luego de ciertas operaciones.

```
function saludar(nombre) {
 console.log("Hola " + nombre);
}

function recibirInvitadoJuan(callback) {
 var invitado = "Juan";
 callback(invitado);
}

recibirInvitadoJuan(saludar); // Hola Juan
```

Su uso en las funciones asíncronas sería el siguiente:

Aunque esta forma a primera vista parece sencilla, puede verse afectada cuando se necesita anidar muchas funciones, a este problema se le llama Callbacks Hell.

```
function saludar(nombre) {
 console.log("Hola " + nombre);
}

function
recibirJuanMasTarde(callback) {
 setTimeout(function () {
   var invitado = "Juan";
   callback(invitado);
 }, 5000);
}

recibirJuanMasTarde(saludar);
// ... (Espera por 5 segundos)
// Hola Juan
```

Promises

Promise o (Promesa) es un objeto en JavaScript que representa un valor que puede o no estar disponible en el futuro. Fueron implementadas para resolver (de una manera mejor estructurada) las tareas asíncronas que se realizaba con los Callbacks.

Las promesas son creadas con el keyword 'new' y pasando un callback que recibe dos parámetros que a su vez también son funciones, que serán llamadas si la operación tuvo éxito (resolve) y si algo falló (reject).

```
function sumarMasTardeMostrar(x, y) {
 var nuevaPromesa = new Promise(function (resolve,
reject) {
   if (!x || !y) {
     reject("Falta un numero");
   }

   setTimeout(function () {
     resolve(x + y);
   }, 1000);
 });
 return nuevaPromesa;
}

sumarMasTardeMostrar(5, 3)
 .then(function (resultado) {
   console.log(resultado);
 })
 .catch(function (error) {
   console.error(error);
 });
```

Luego que tenemos esta función asíncrona, utilizamos then cuando obtenemos el resultado y catch para capturar los errores.

Las ventajas de las promises es que ya muchas de las nuevas funcionalidades asíncronas que se le brinda al lenguaje, retornan una promise por defecto. Este es el caso de fetch, que es provisto por los navegadores para hacer peticiones HTTP. Aquí vemos que además de fetch también Body.json utiliza promise. Este último es el objeto resultante de la petición, que a su vez utiliza este método para convertir la respuesta objeto JSON.

```
const url = "https://example.com/api/users";
const body = {
 //   ...
};

fetch(url, body)
 .then(function (response) {
   return response.json();
 })
 .then(function (jsonResponse) {
   return console.log(jsonResponse);
 })
 .catch(function (error) {
   console.error(error);
 });
```

Async - Await

Por último, tenemos este método, introducido como standard en la especificación de ECMA 2017. Al igual que then y catch éstas trabajan con Promise, solo que de una manera muy particular y haciendo que la sintaxis sea más legible.

Lo primero que debemos hacer crear una función con el keyword async delante. Así podremos utilizar el await dentro de esta función. El await debe ser utilizado delante de la sentencia que nos devolverá la promesa. Así en vez de obtener el objeto Promise, obtendremos el valor esperado.

```
function async getUsers () {
  const url = "https://example.com/api/users"
  const body = {
  //   ...
  }

  const response = await fetch(url, body)
  const jsonResponse = await response.json()
}
```

Debemos tomar en cuenta que JavaScript tiene funciones que no retornarán su valor de forma inmediata y aprender a utilizar este comportamiento a nuestro favor.

Manejar este conocimiento nos da muchas alternativas y la posibilidad de utilizar todas las capacidades del lenguaje.

MÓDULO 3

GRACIAS

Aquí finaliza la clase n°6 del módulo 3

OTEC PLATAFORMA 5 CHILE