****

**UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE**

FACULTAD DE CIENCIAS DE LA COMPUTACION

PROGRAMACIÓN WEB

NRC: 2260

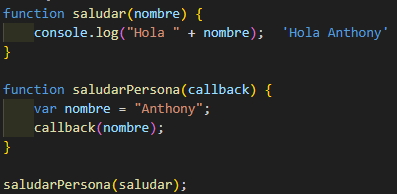
NOMBRE: ANTHONY MORALES

FECHA: 17/01/2025

**FUNCION CALLBACK**

Una función callback o *retrollamadas* es una función que se pasa como argumento a otra función, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

**Ejemplo:**

****

El ejemplo anterior es un callback sincrónico, ya que se ejecuta inmediatamente.

Sin embargo, hay que tener en cuenta que los callbacks se utilizan para continuar con la ejecución del código después de que se haya completado una operación a sincrónica.

**SETINTERVAL**

El método setinterval(), llama a una función o ejecuta un fragmento de código de forma reiterada, con un retardo de tiempo fijo entre cada llamada.

**Sintaxis:**

setInterval(code);

setInterval(code, delay);

setInterval(func);

setInterval(func, delay, arg0, arg1);

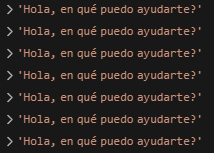
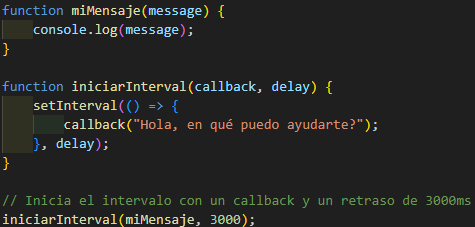
**Parámetros:**

func: Una función que se ejecutara cada cierto tiempo, el tiempo lo determina delay, estando en milisegundos. La primera ejecución ocurre tras el tiempo determinado por delay.

code: Una sintaxis especial que permite incluir una cadena en lugar de una función, la cual es compilada y ejecutada cada delay milisegundos. Se recomienda no usar esta sintaxis, por la misma razón que hace que el uso de eval()sea un riesgo de seguridad.

delay: El tiempo en milisegundos (milésimas de segundo) que el temporizador debe retrasar cada ejecución de la función o código especificado.

arg0, arg1: Argumentos adicionales que se pasan a la función especificada por *func* una vez que el temporizador expira.



En este ejemplo se usa como callback **miMensaje,** mismo que pasa como argumento a **iniciarInterval.** 3000 es el número de milisegundos. Por ende, se llamará después de 3 segundos.

**SETTIMEOUT**

El método settimeout(), establece un temporizador que ejecuta una función o una pieza de código especifica una vez que expira el temporizador.

**Sintaxis:**

setTimeout(functionRef)

setTimeout(functionRef, delay)

setTimeout(functionRef, delay, param1)

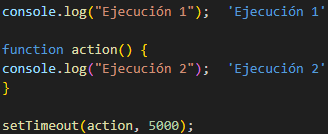
setTimeout(functionRef, delay, param1, param2)

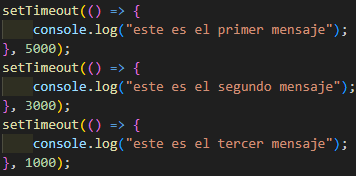
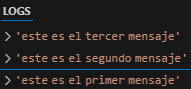
**Parametros:**

functionRef: Una función que se ejecutará después de que expire el temporizador.

code: Una sintaxis alternativa que le permite incluir una cadena en lugar de una función, que se compila y ejecuta cuando expira el temporizador. Aunque esta sintaxis no se recomiendo.

delay: El tiempo, en milisegundos, que el temporizador debe esperar antes de que se ejecute la función o el código especificado. Si se omite este parámetro, se utiliza un valor de 0, lo que significa ejecutar "inmediatamente".

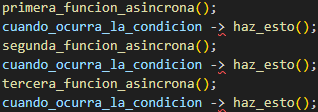


**PROMESAS**

Una *Promesa* (**Promise**) es un proxy de un valor que no necesariamente se conoce cuando se crea la promesa. Le permite asociar controladores con el valor eventual de éxito o el motivo de falla de una acción asíncrona. Es algo que en principio pensamos que se cumplirá en el futuro, pero a medida que pasa el tiempo pueden ocurrir varias cosas.

Es como una especie de condición que «se queda en espera» y se ejecutará en el futuro. De esta forma, podemos manejar código asíncrono sin bloquear la ejecución del resto del programa.

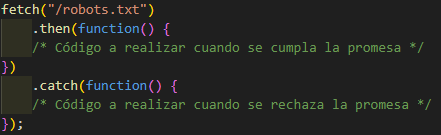


Al ejecutar la función asíncrona, el programa continua con la siguiente línea de código. Realmente, no ha terminado de ejecutar el código de la función anterior, pero se ha quedado en espera, hasta que se cumpla el criterio o condición de la promesa.

Podemos trabajar con promesas de varias formas. Una de ellas es usar los **thennables**, donde se utiliza un método llamado .then()

|  |  |
| --- | --- |
| Método | Descripción |
| .then(function resolve) | Ejecuta la función cuando la promesa se cumple. |
| .catch(function reject) | Ejecuta la función cuando la promesa se rechaza. |
| .then(function resolve, function reject) | Equivale a las 2 anteriores en el mismo .then() |
| .finally(function end) | Ejecuta la función cuando sale de pendiente. |

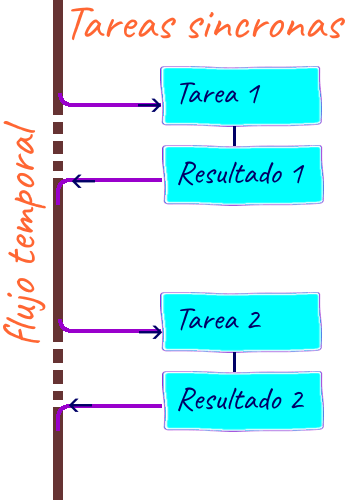
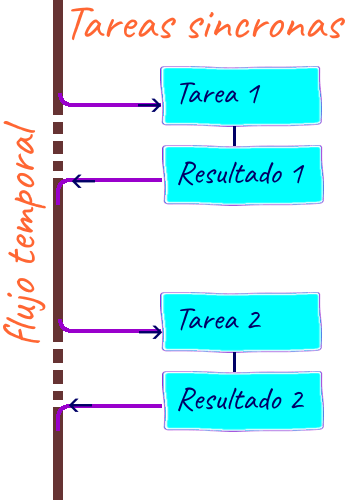
La función fetch() devuelve una promesa que se cumple cuando el navegador obtiene la información solicitada. catch() que al contrario que el método .then(), este método ejecutará la función callback pasada por parámetro cuando la **promesa se rechaza.**



**PROCESOS SÍNCRONOS Y ASÍNCRONOS**

En resumen, cuando en la ejecución de un programa cada tarea se ejecuta después de que ha terminado otra tarea anterior estamos hablando de **procesos** **síncronos**.

Cuando en un proceso una tarea no tiene que esperar a que se termine de ejecutar otra tarea para continuar estamos hablando de **procesos** **asíncronos**.

Sin embargo, hay que tener en cuenta que las tareas asíncronas son útiles cuando hay partes del programa que pueden tardar un tiempo indeterminado en ser procesadas y no son imprescindibles para continuar con la ejecución del script. Como puede ser el uso del método **setTimeout()**, como se demuestra en el siguiente ejemplo:

