

ACÀMICA

# Agenda

---

## Daily

Programo: Async / Await

Buenas prácticas

## Break

Programamos: Open Movie Database

Programan: Async / Await

## Cierre



# API

Una API (Application Programming Interface) es una serie de reglas y especificaciones para que las aplicaciones puedan comunicarse entre ellas.

# Daily



Daily



## Sincronizando...

### Toolbox



¿Cómo te ha ido?  
¿Obstáculos?  
¿Cómo seguimos?

### Challenge



¿Cómo te ha ido?  
¿Obstáculos?  
¿Cómo seguimos?

# Funciones Async



## Funciones Async

Las funciones async son funciones que permiten escribir código basado en promesas de tal manera que se comporte como **código sincrónico**, pero sin bloquear el hilo principal de ejecución.



## Async / Await

**Async** es la palabra reservada que se antepone a **function**. Sirve para definir una función asíncrona con la posibilidad de detener el hilo de ejecución mediante **await**.





# Async / Await




The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the execution of a JavaScript function `getInfoFromGitHub` which uses `fetch` to retrieve user data from the GitHub API. The function is called with the username `'frenchita'`. The output is a JSON object representing the user's profile information.

```
> function getInfoFromGitHub(username){  
  const endpoint = `https://api.github.com/users/${username}`  
  fetch(endpoint)  
    .then(respuesta => respuesta.json())  
    .then(usuario => console.log(usuario))  
}  
  
getInfoFromGitHub('frenchita');  
  
< undefined  
  
{login: "frenchita", id: 54448602, node_id: "MDQ6VXNlcjU0NDQ4NjAy", avatar_url:  
  r_id: "", ...}  
  avatar_url: "https://avatars3.githubusercontent.com/u/54448602?v=4"  
  bio: null  
  blog: ""  
  company: null  
  created_at: "2018-08-23T16:11:42Z"
```

Iniciemos con una función escrita en promesas para extraer la información de una persona usuaria a través de la API de GitHub.

# Async / Await



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its output:

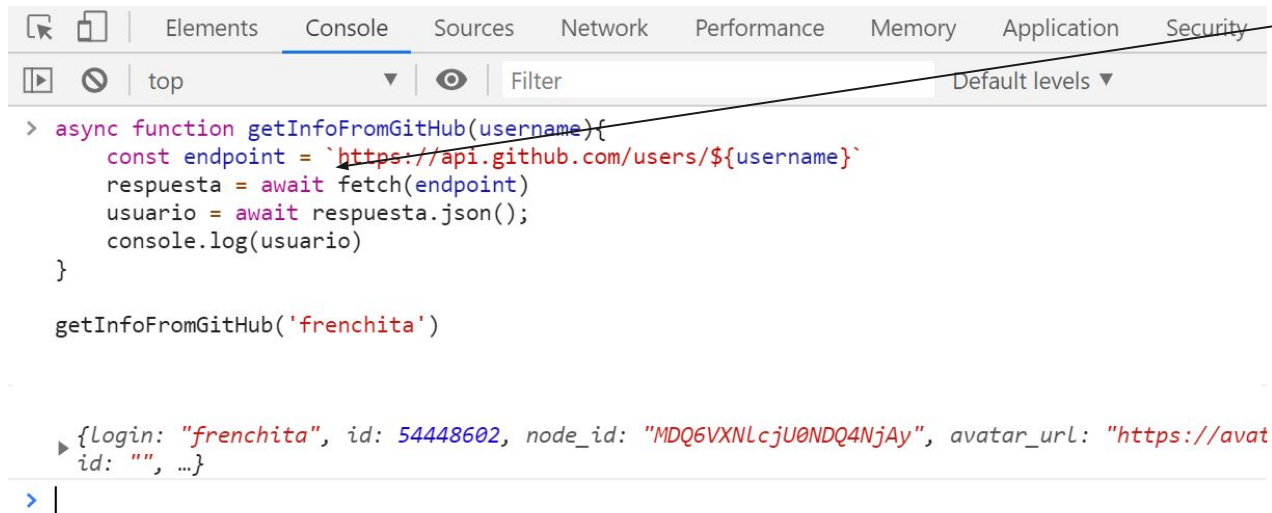
```
> async function getInfoFromGitHub(username){  
  const endpoint = `https://api.github.com/users/${username}`  
  fetch(endpoint)  
    .then(respuesta => respuesta.json())  
    .then(usuario => console.log(usuario))  
}  
  
getInfoFromGitHub('frenchita')
```

The output of the function is a JSON object:

```
{login: "frenchita", id: 54448602, node_id: "MDQ6VXNlcjU0NDQ4NjAy", avatar_url: "https://avc  
id: "", ...}
```

A través de la palabra reservada **async** vamos a hacer que nuestra función se transforme en una función tipo síncrona.

# Async / Await

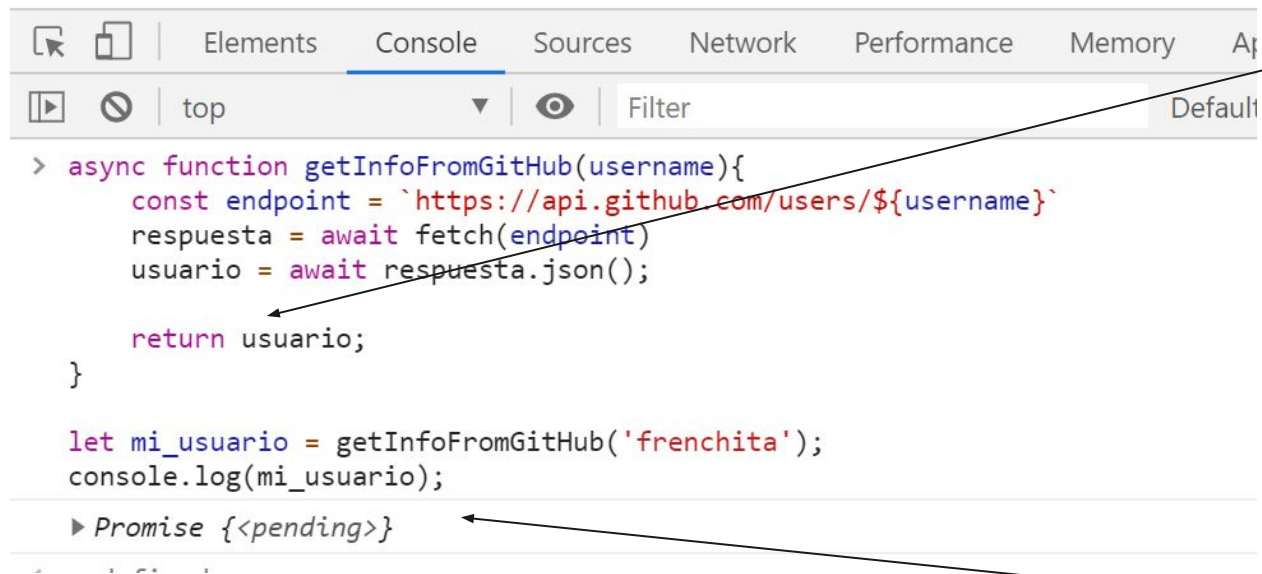


```
> async function getInfoFromGitHub(username){  
  const endpoint = `https://api.github.com/users/${username}`  
  respuesta = await fetch(endpoint)  
  usuario = await respuesta.json();  
  console.log(usuario)  
}  
  
getInfoFromGitHub('frenchita')  
  
{login: "frenchita", id: 54448602, node_id: "MDQ6VXNLCjU0NDQ4NjAy", avatar_url: "https://avat  
id: "", ...}  
> |
```

Mediante la palabra reservada **await** hacemos que el flujo de la aplicación se detenga hasta que finalice la instrucción.

Ya hemos reemplazado la promesa que teníamos dentro de la función.

# Async / Await



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its execution result:

```
> async function getInfoFromGitHub(username){  
  const endpoint = `https://api.github.com/users/${username}`;  
  respuesta = await fetch(endpoint);  
  usuario = await respuesta.json();  
  
  return usuario;  
}  
  
let mi_usuario = getInfoFromGitHub('frenchita');  
console.log(mi_usuario);
```

The execution result is shown as:

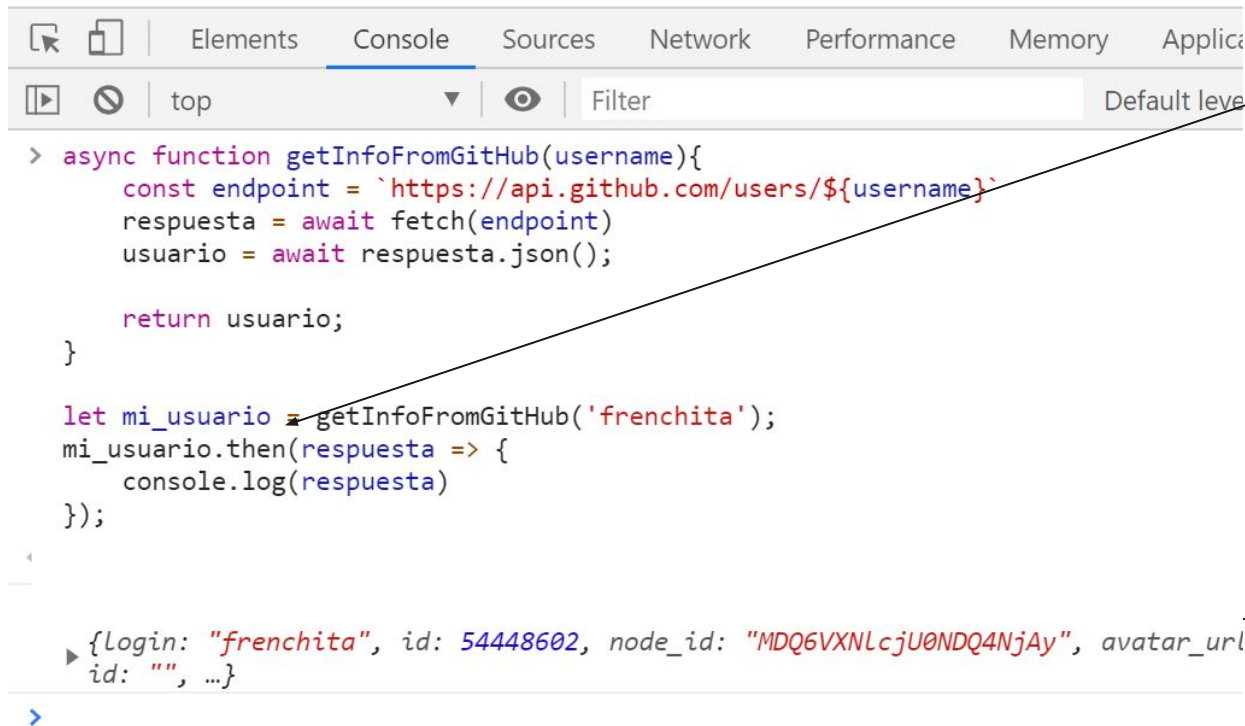
```
► Promise {<pending>}
```

Two arrows originate from the text on the right. One arrow points from the word 'fetch' in the text to the `fetch(endpoint)` call in the code. The other arrow points from the text 'Las funciones tipo async siempre retornan una promesa.' to the `Promise {<pending>}` result in the console.

Ahora vamos a tener retornar la información que obtuvimos de la llamada **fetch**.

Las funciones tipo **async** siempre retornan una promesa.

# Async / Await



```
> async function getInfoFromGitHub(username){
  const endpoint = `https://api.github.com/users/${username}`
  respuesta = await fetch(endpoint)
  usuario = await respuesta.json();

  return usuario;
}

let mi_usuario = getInfoFromGitHub('frenchita');
mi_usuario.then(respuesta => {
  console.log(respuesta)
});
```

```
{login: "frenchita", id: 54448602, node_id: "MDQ6VXNLCjU0NDQ4NjAy", avatar_url:
id: "", ...}
```

Como la respuesta de la función es una promesa, accede a la información mediante el método `.then`.

# Async / Await



```
> async function getInfoFromGitHub(username){
  const endpoint = `https://api.github.com/users/${username}`
  respuesta = await fetch(endpoint)
  usuario = await respuesta.json();

  if(respuesta.status !== 200)
    throw Error(`Usuario ${username} inexistente`)

  return usuario;
}

let mi_usuario = getInfoFromGitHub('xzcvzasdfasdfcv');
mi_usuario
  .then(respuesta => console.log(respuesta))
  .catch(error => console.log(error));
```

✖ ▶ GET https://api.github.com/users/xzcvzasdfasdfcv 404 (Not Found)

Error: Usuario xzcvzasdfasdfcv inexistente  
at getInfoFromGitHub (<anonymous>:7:15)

> |

Como nuestra función retorna una promesa, podemos dar aviso de cualquier error que suceda y capturarlo en nuestro hilo principal mediante el método `.catch`.

# Async / Await

Ahora bien, eliminamos el uso de promesas dentro de nuestra función, pero necesitamos utilizarlas para obtener el contenido en nuestro hilo principal.

# Async / Await

Ahora bien, eliminamos el uso de promesas dentro de nuestra función, pero necesitamos utilizarlas para obtener el contenido en nuestro hilo principal.

Para evitar utilizar promesas en nuestro hilo principal, vamos a crear una función que se ejecute automáticamente al momento de crearse, y mediante **try / catch** realizaremos nuestras interacciones.

```
(async function () {  
    //función que se ejecuta al momento de crearla  
}) ();
```



# Async / Await

```
> async function getInfoFromGitHub(username){  
  const endpoint = `https://api.github.com/users/${username}`  
  respuesta = await fetch(endpoint)  
  usuario = await respuesta.json();  
  if(respuesta.status !== 200)  
    throw Error(`Usuario ${username} inexistente`)  
  return usuario;  
}
```

```
(async function (){  
  try {  
    let respuesta = await getInfoFromGitHub('frenchita');  
    console.log(respuesta);  
  } catch (error) { console.log(error) }  
})();
```

```
< ▶ Promise {<pending>}
```

```
▶ {login: "frenchita", id: 54448602, node_id: "MDQ6VXNlcjU0NDQ4NjAy", avatar_id: "", ...}
```

```
>
```

La función tipo `async` que se ejecuta a sí misma al momento de crearse llama a `getInfoFromGitHub` con un `await`, y en la siguiente línea ya tenemos la información disponible. Si la llamada a la API da un status diferente a 200 una excepción es lanzada y capturada por el `.catch`.

# Programo

squad leads



# Funciones tipo Async

Veamos una demostración de cómo trabajar con estas funciones.



# Buenas prácticas



**Await** nos permite detener el hilo de ejecución en una función tipo **Async**, pero...

**¡NO** detengas la ejecución por cada instrucción que realices así no generas tiempos de espera innecesarios!

```
async function doSomething() {  
  
    await .....  
    await .....  
    await .....  
    await .....  
    await .....  
    await .....  
    return something;  
}
```



A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver fork are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

**¡BREAK!**

---



# Programamos

todos/as



Javascript

## Programamos

Programemos la siguiente estructura:

The Open Movie Database es una plataforma que ofrece una API para obtener información sobre películas. Vamos a registrarnos y realizar algunas llamadas a su API:

<http://www.omdbapi.com/>



Javascript

# Programamos

Programemos la siguiente estructura:

Cuando obtengas tu API KEY, haz un request para obtener información sobre tu película favorita.

Crea una card con la información de la película.



Card Title

I am a very simple card. I am good at containing small bits of information. I am convenient because I require little markup to use effectively.

# Programman

trainees



Javascript

# Programan

Crea una carpeta estática en tu proyecto para generar HTMLs.

Allí, usa `fetch` para realizar un llamado a tu API para que sea ella la encargada de buscar la información en Open Movie Database y mostrar el resultado en una página web.



## Card Title

I am a very simple card. I am good at containing small bits of information. I am convenient because I require little markup to use effectively.

**Sprint 2 (Meeting #20)**

Nos tomamos unos minutos para completar [esta encuesta](#).

Queremos saber cómo valoran mi tarea hasta acá. ¡Les va a llevar solo un minuto!



# Para la próxima

---

- 1) Termina el ejercicio de la meeting de hoy.
- 2) Lee la toolbox 21.
- 3) Resuelve el challenge.

ACÀMICA