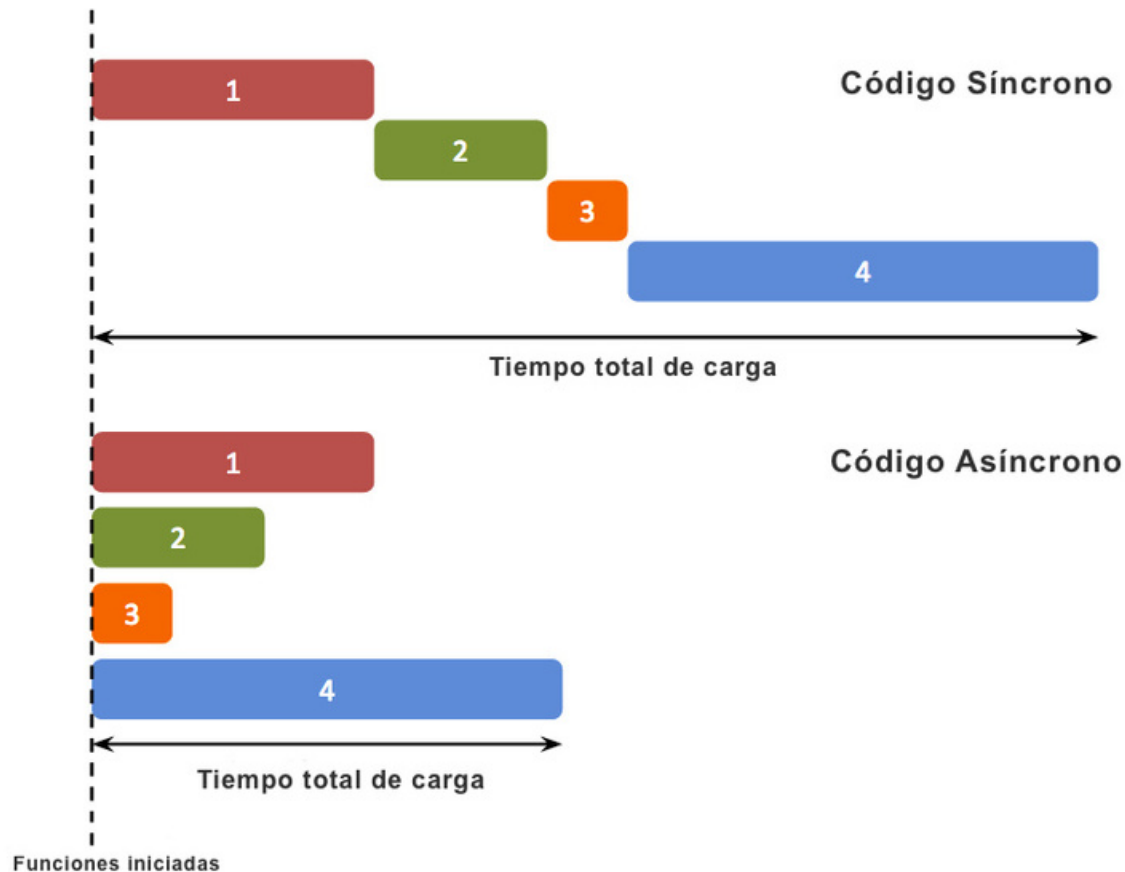
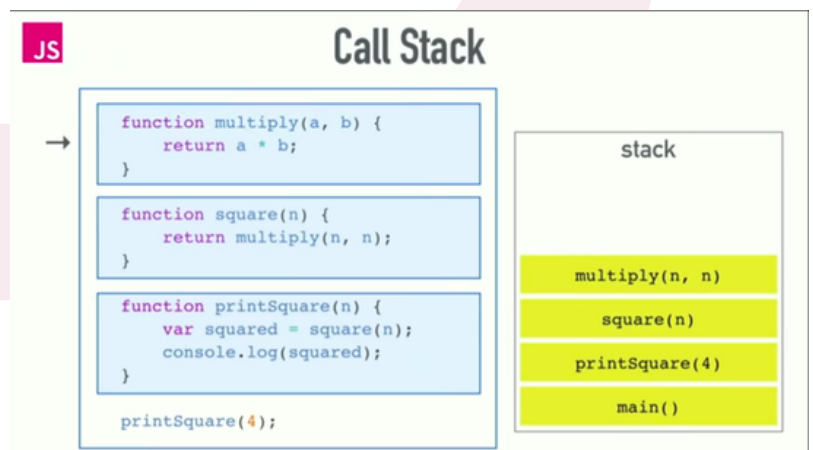


Sincronismo y Asincronismo



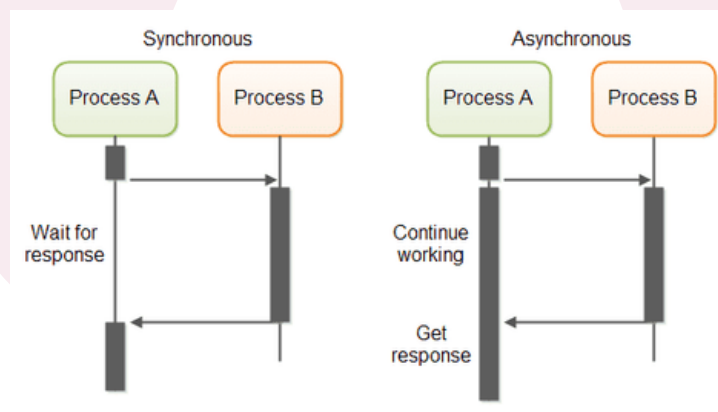
Sincronismo:

- En el sincronismo, las tareas se ejecutan de manera secuencial, una tras otra, siguiendo el orden en que fueron escritas en el código.
- Cada vez que una tarea se ejecuta, se agrega a la pila de ejecución, y el programa espera a que esta tarea se complete antes de pasar a la siguiente.
- La pila de ejecución es como una pila de platos o libros: la tarea actual está en la parte superior y se ejecuta, y una vez que se completa, se retira de la pila y se pasa a la siguiente tarea.



Asincronismo:

- En el asincronismo, las tareas pueden ejecutarse en cualquier orden y no necesariamente de manera secuencial.
- Cuando una tarea asincrónica se ejecuta, no bloquea el flujo del programa. En su lugar, se agrega a la pila de ejecución y se maneja de manera diferida.
- Por ejemplo, si se hace una solicitud a un servidor para obtener datos, el código no espera bloqueado a que lleguen los datos. En su lugar, la solicitud se envía y el programa sigue ejecutando otras tareas. Cuando los datos estén listos, se manejarán en un callback o se ejecutará una función de promesa.
- Mientras tanto, el programa continúa su ejecución, manejando otras tareas o eventos.
- Es importante recordar que JavaScript es un lenguaje de programación de un solo subproceso (single-threaded), lo que significa que solo puede ejecutar una tarea a la vez. Sin embargo, el asincronismo permite que el programa maneje múltiples tareas de manera efectiva gracias al uso de callbacks, promesas o async/await.
- Un ejemplo sobre código asincrónico son los eventos del dom (onClick, onLoad, etc), setTimeout o setInterval y las promesas



En resumen, el sincronismo y el asincronismo en JavaScript se refieren a cómo se manejan las tareas en el flujo de ejecución del programa, ya sea ejecutándolas secuencialmente en la pila de ejecución o manejándolas de manera diferida mientras el programa continúa su ejecución.

VIDEO RECOMENDADISIMO DE VER:
[CLICK AQUI](#)

PROMESAS

- Una promesa es un objeto que representa el resultado de una operación asíncrona, que puede ser exitosa (resuelta) o fallida (rechazada) en algún momento en el futuro.
- Las promesas tienen tres estados: pendiente (pending), resuelta (fulfilled) y rechazada (rejected).
- Se pueden crear nuevas promesas utilizando el constructor Promise, pasándole una función con dos parámetros: resolve y reject. Dentro de esta función, se realiza la operación asíncrona y se llama a resolve cuando la operación se completa exitosamente, o a reject si ocurre algún error.

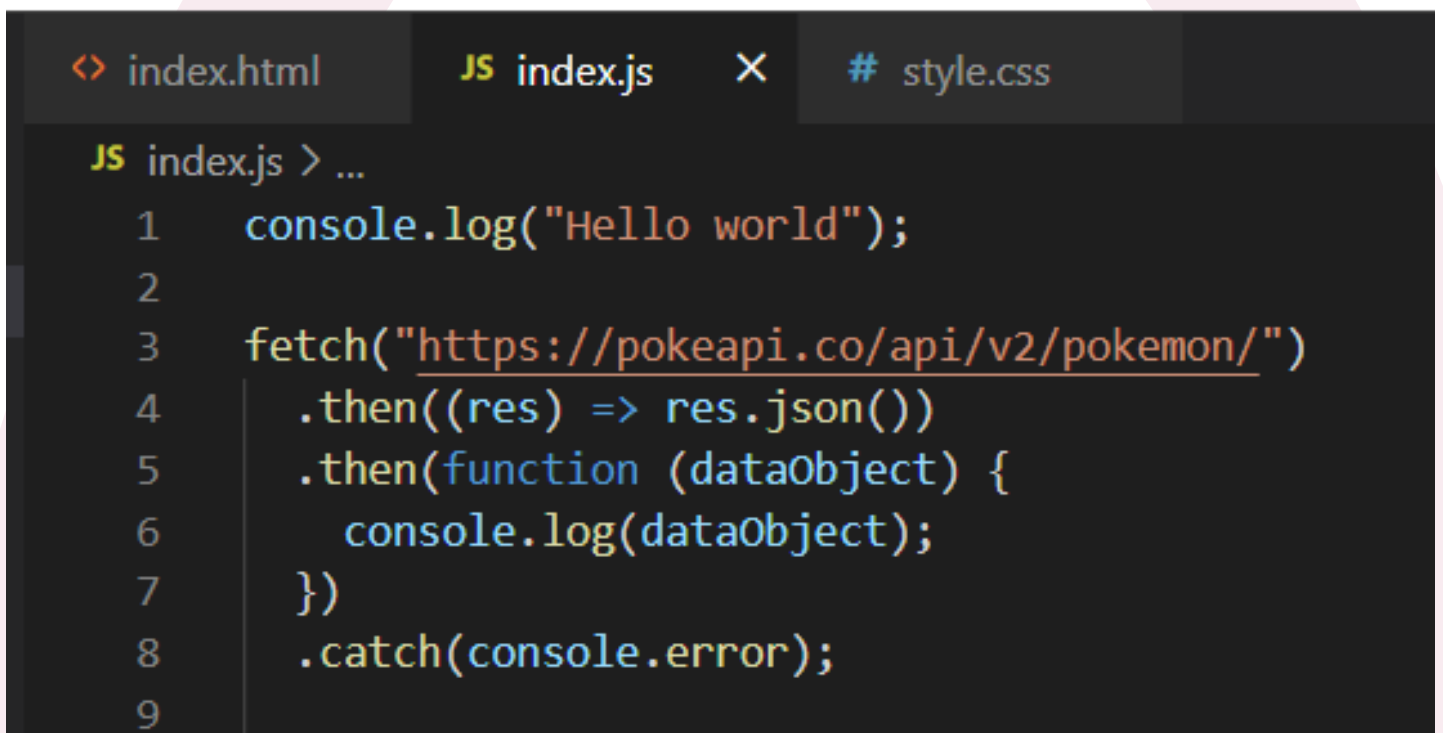
```
1  let promiseToMowLawn = new Promise(function(resolve, reject) {
2
3      // mowing the lawn
4      let isLawnMowed = true;
5
6      if (isLawnMowed) {
7          resolve('mowed');
8      } else {
9          reject();
10     };
11
12     });
13
14     promiseToMowLawn.then(function(fromResolve) {
15         console.log('The lawn is ' + fromResolve);
16     });
```

- Una promesa es un objeto que representa el resultado de una operación asíncrona, que puede ser exitosa (resuelta) o fallida (rechazada) en algún momento en el futuro.
- Las promesas tienen tres estados: pendiente (pending), resuelta (fulfilled) y rechazada (rejected).
- Se pueden crear nuevas promesas utilizando el constructor Promise, pasándole una función con dos parámetros: resolve y reject. Dentro de esta función, se realiza la operación asíncrona y se llama a resolve cuando la operación se completa exitosamente, o a reject si ocurre algún error.

PARA MANEJAR LAS PROMESAS TENEMOS DOS FORMAS

.then/.catch:

- .then y .catch son métodos que se utilizan con promesas en JavaScript.
- .then se usa para manejar el resultado exitoso de una promesa, es decir, cuando la promesa se resuelve correctamente.
- .catch se usa para manejar errores que puedan ocurrir durante la ejecución de una promesa.
- .finally (Opcional) se utiliza para ejecutar código siempre al finalizar la promesa independientemente de su resultado



The screenshot shows a code editor with three tabs: 'index.html', 'JS index.js', and 'style.css'. The 'JS index.js' tab is active, displaying the following code:

```
JS index.js > ...
1  console.log("Hello world");
2
3  fetch("https://pokeapi.co/api/v2/pokemon/")
4    .then((res) => res.json())
5    .then(function (dataObject) {
6      console.log(dataObject);
7    })
8    .catch(console.error);
9
```

async/await:

- async es una palabra clave que se usa para declarar una función asíncrona, que permite el uso de await.
- await se usa dentro de una función async para esperar la resolución de una promesa de forma síncrona.
- Hace que el código parezca síncrono, aunque internamente se sigue ejecutando de forma asíncrona.
- Facilita la escritura de código más legible y comprensible.
- Evita Callbacks Hell

```

async function boom() {
  try {
    const fireworks = await getFireworks();
    const trigger = await setUpFireworks(fireworks);
    return trigger();
  } catch(error) {
    return 'Run Away!'
  }
}

```

EVITAR SIEMPRE CALLBACKS HELL

```

1 function hell(win) {
2   // for listener purpose
3   return function() {
4     loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5       loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6         loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7           loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8             loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9               loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10                loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                  loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                    loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                      async.eachSeries(SERIALS, function(src, callback) {
14                        loadScript(win, BASE_URL+src, callback);
15                      });
16                    });
17                  });
18                });
19              });
20            });
21          });
22        });
23      });
24    });
25  });
26 }

```

```

pan.pourWater(function() {
  range.bringToBoil(function() {
    range.lowerHeat(function() {
      pan.addRice(function() {
        setTimeout(function() {
          range.turnOff();
          serve();
        }, 15 * 60 * 1000);
      });
    });
  });
});

```

pyramid of doom