

HENRY

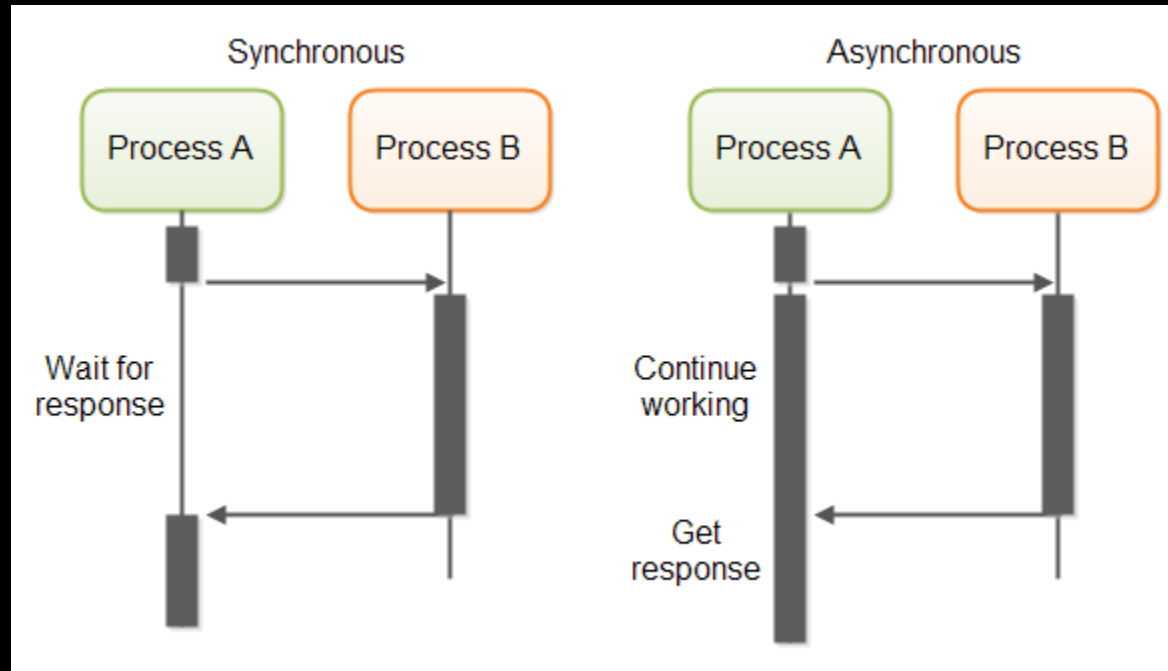
A bright yellow beam of light originates from the left edge of the frame and points towards the letter 'R' in the word 'HENRY'. The beam is wider on the left and tapers as it moves to the right. The word 'HENRY' is written in a bold, black, sans-serif font.



Promises



Promises





¿Qué es una Promise?

“ Una promesa representa el eventual resultado de una operación asíncrona.

- The Promise A+ Spec



Las Promesas son Objetos

Es un objeto que **representa** y **gestiona** el lifecycle de una respuesta futura.

El objeto mantiene dentro suyo:

status(pending, **fulfilled**, **rejected**)
information (**value** or a **reason**)

Sólo podemos acceder al método:

.then()



Las Promesas son Objetos

El estado de las promesas sólo cambia una vez.

```
new Promise(executor)
```

```
state: "pending"  
result: undefined
```

resolve(value)

```
state: "fulfilled"  
result: value
```

reject(error)

```
state: "rejected"  
result: error
```

```
unaPromise.then(succesfullHandler, errorHandler)
```



Promises

Un vez que una promesa se fullfilea, o es rechazada.
Ejecuta sus handlers

Si le agregamos un handler nuevo, lo ejecuta con el mismo
valor al que se fullfileó.

```
1 aPromise.then(handler, errorHandler);
2
3 // se fullfilea la promesa, por ejemplo con un valor (sin error);
4 // se ejecuta el handler.
5
6 aPromise.then(handler2);
7
8 // si agregamos un nuevo handler, se ejecuta con el mismo valor!
```

Creando Promesas

```
1
2 var promise = new Promise(function(resolve, reject) {
3   // Hacer cosas acá dentro, probablemente asincrónicas.
4
5   if (/* Todo funcionó como esperabamos*/) {
6     resolve("Jooya!");
7   }
8   else {
9     reject(Error("Algo se rompió"));
10  }
11 });
```




Callback Hell

```
1 const verifyUser = function(username, password, callback){
2   dataBase.verifyUser(username, password, (error, userInfo) => {
3     if (error) {
4       callback(error)
5     }else{
6       dataBase.getRoles(username, (error, roles) => {
7         if (error){
8           callback(error)
9         }else {
10          dataBase.logAccess(username, (error) => {
11            if (error){
12              callback(error);
13            }else{
14              callback(null, userInfo, roles);
15            }
16          })
17        }
18      })
19    }
20  })
21 };
```



Callback Hell

```
1  const verifyUser = function(username, password) {  
2    database.verifyUser(username, password)  
3      .then(userInfo => dataBase.getRoles(userInfo))  
4      .then(rolesInfo => dataBase.logAccess(rolesInfo))  
5      .then(finalResult => {  
6        //do whatever the 'callback' would do  
7      })  
8      .catch((err) => {  
9        //do whatever the error handler needs  
10     });  
11  };
```



Encadenado Promesas

.then() retorna una promesa! por eso las podemos encadenar!

```
1 var primerMetodo = function() {
2   var promise = new Promise(function(resolve, reject){
3     setTimeout(function() {
4       console.log('Terminó el primer método');
5       resolve({num: '123'}); //pasamos unos datos para ver como los manejamos
6     }, 2000); // para simular algo asincronico hacemos un setTimeout de 2 s
7   });
8   return promise;
9 };
10
11
12 var segundoMetodo = function(datos) {
13   var promise = new Promise(function(resolve, reject){
14     setTimeout(function() {
15       console.log('Terminó el segundo método');
16       resolve({nuevosDatos: datos.num + ' concatenamos texto y lo pasamos'});
17     }, 2000);
18   });
19   return promise;
20 };
21
22 var tercerMetodo = function(datos) {
23   var promise = new Promise(function(resolve, reject){
24     setTimeout(function() {
25       console.log('Terminó el tercer método');
26       console.log(datos.nuevosDatos); //imprimos los datos concatenados
27       resolve('hola');
28     }, 3000);
29   });
30   return promise;
31 };
32
33 primerMetodo()
34   .then(segundoMetodo)
35   .then(tercerMetodo)
36   .then(function(datos){
37     console.log(datos); //debería ser el 'hola' que pasamos en tercerMetodo
38   });
```

< DEMO />