

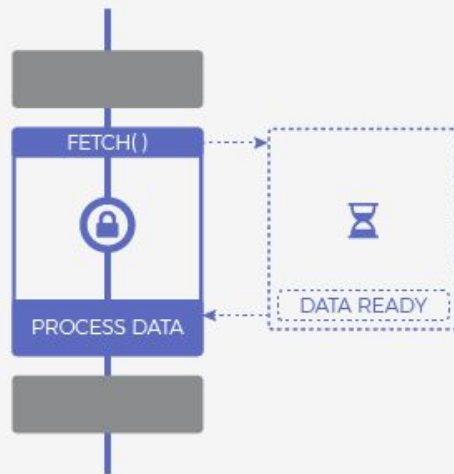
A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is light green. They are positioned diagonally, with the blue one partially covering the green one.

Asincronismo js



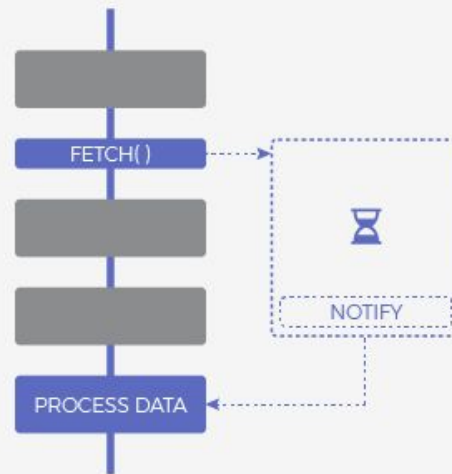
Significado

- Ejecución de tareas no secuenciales, notificando de la finalización de la tarea y tratando dicha respuesta




SÍNCRONO

Ejecución secuencial. Retorna cuando la operación ha sido completada en su totalidad.



ASÍNCRONO

La finalización de la operación es notificada al programa principal. El procesado de la respuesta se hará en algún momento futuro.



Patrones asíncronos

- Callbacks
- Promesas
- Async / Await
- Observables (RxJS)




Callbacks

- Función pasado como argumento de otra función

```
setTimeout(myFunction, 3000);
```

```
function myFunction() {  
  console.log("Función llamada");  
}
```



```
function tratarJson(datos)
{
    console.log(datos);
}
```

```
cargarJson(tratarJson);
```

```
function cargarJson(callback)
{
    let req = new XMLHttpRequest();
    req.open("GET", "posts.json");

    req.onload = function ()
    {
        if (req.status == 200)
        {
            callback(JSON.parse(req.responseText));
        }
        else
        {
            callback("Error: " + req.status);
        }
    }

    req.send();
}
```



Promesas

- Objeto que representa la finalización de una tarea asíncrona, ya sea exitosa o fracasada
- Basadas en callbacks, resuelven el mismo problema de una forma más legible
- Debemos distinguir dos partes importantes:
 - Como crear promesas
 - Como consumir promesas

PROMESAS



PROMESA PENDIENTE



PROMESA CUMPLIDA



PROMESA RECHAZADA





Métodos	Descripción
<code>.then(<small>FUNCTION</small> resolve)</code>	Ejecuta la función callback resolve cuando la promesa se cumple.
<code>.catch(<small>FUNCTION</small> reject)</code>	Ejecuta la función callback reject cuando la promesa se rechaza.
<code>.then(<small>FUNCTION</small> resolve, <small>FUNCTION</small> reject)</code>	Método equivalente a las dos anteriores en el mismo <code>.then()</code> .
<code>.finally(<small>FUNCTION</small> end)</code>	Ejecuta la función callback end tanto si se cumple como si se rechaza.



```
function tratarJson(datos)
```

```
{
```

```
    console.log(datos);
```

```
}
```

```
cargarJson()
```

```
.then(r =>
```

```
{
```

```
    tratarJson(r);
```

```
})
```

```
.catch(e => {
```

```
    tratarJson(e);
```

```
});
```

```
function cargarJson()
```

```
{
```

```
//return new Promise((resolve, reject) => { return new Promise(function(resolve, reject)
```

```
{
```

```
    let req = new XMLHttpRequest();
```

```
    req.open("GET", "posts.json");
```

```
    req.onload = function ()
```

```
{
```

```
    if (req.status == 200)
```

```
{
```

```
        resolve(JSON.parse(req.response));
```

```
}
```

```
    else
```

```
{
```

```
        reject("Error: " + req.status);
```


```
}
```

```
}
```

```
    req.send();
```

```
})
```

```
}
```



```
fetch('posts.json')
  .then(function(respuesta)
  {
    return respuesta.json();
  })
  .then(function(datos)
  {
    console.log(datos);
  })
  .catch(function(err)
  {
    console.error(err);
  });
```

```
fetch('posts.json')
  .then(respuesta => respuesta.json())
  .then(datos=> console.log(datos))
  .catch(function(err)
  {
    console.error(err);
  });
```



```
var p1 = new Promise(function(resolve, reject)
{
    setTimeout(resolve, 5000, "Primera")
})
```

```
var p2 = new Promise(function(resolve, reject)
{
    setTimeout(resolve, 1000, "Segunda")
})
```

```
Promise.all([p1, p2]).then(function(value)
{
    console.log(value)
})
```

```
Promise.race([p1, p2]).then(function(value)
{
    console.log(value)
})
```



Async / Await

- Palabras reservadas con comportamiento específicos
 - Async: Modificador de función que determina que dicha función debe devolver una promesa
 - Await: Espera la ejecución de la promesa de la función asíncrona y una vez terminada le asigna el valor. Solo puede usarse dentro de una función async



```
async function loadJson(url)
{
    let response = await fetch(url);

    if (response.status == 200)
    {
        let json = await response.json();
        return json;
    }

    throw new Error(response.status);
}
```

```
loadJson('posts.json')
    .then(j => {
        console.log(j);
    })
    .catch(e => {
        console.log(e);
    });
```