# Asynchrone

## API

Une API (Application Programming Interface ou Interface de Programmation Applicative en français) est une interface, c'est-à-dire un ensemble de codes grâce auxquels un logiciel fournit des services à des clients.
Le principe et l'intérêt principal d'une API est de permettre à des personnes externes de pouvoir réaliser des opérations complexes et cachant justement cette complexité.
En effet, en tant que développeur nous n'aurons pas besoin de connaître les détails de la logique interne du logiciel tiers et n'y aura d'ailleurs pas accès directement puisque nous devrons justement passer par l'API qui va nous fournir en JavaScript un ensemble d'objets et donc de propriétés et de méthodes prêtes à l'emploi et nous permettant de réaliser des opérations complexes.

Il existe des API pour à peu près tout, les plus classiques que vous connaissez sont par exemple les API Google Maps, la météo, l'API Geolocation qui va nous permettre de définir des données de géolocalisation ou encore l'API Canvas qui permet de dessiner et de manipuler des graphiques dans une page.
Nous allons donc organiser notre code de manière à pouvoir contacter une API qui va nous renvoyer une réponse contenant des données, il existe plusieurs types d'API qui renvoi plusieurs types de réponse.

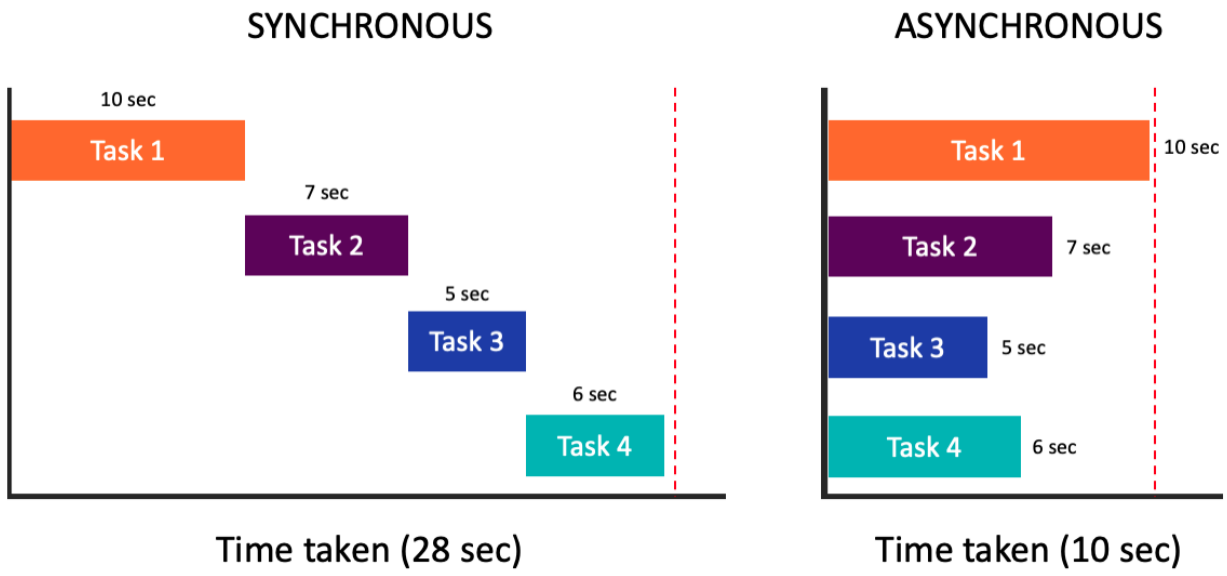
Actuellement la plupart des API sont des API restful c'est-à-dire que le format des réponses que renvoi l'API peut être en JSON, HTML, XLT,Python,PHP ou simplement du texte brut.
Désormais, beaucoup d'API vont renvoyer des réponses au format JSON (Javascript Object Notation), une notation d'objet en Javascript. (il existe une méthode native de JS pour transformer des objets JSON, la méthode .json())
Dans les faits techniquement pour nous une API ça sera simplement une URL que l'on contactera via Javascript pour en extraire les informations.

## Fetch( )

Dans Javascript, nous allons utiliser la méthode fetch(), qui nous permettra de contacter n'importe qu'elle API via son URL, la méthode renvoi des objets de type Response ou Promise.
l'API Fetch et sa méthode fetch() qui correspondent à la "nouvelle façon" d'effectuer des requêtes HTTP.

## Code Asynchrone

Un autre concept essentiel lorsque nous allons contacter des API, c'est d'organiser notre code de manière asynchrone.
L'idée c'est que de base nos programme Javascript sont en mode synchrone, à savoir que chaque ligne de code qui représente une instruction, celle-ci se termine ET seulement ensuite JS passe à l'exécution de la ligne de code suivante.
À la différence d'un code asynchrone qui va permettre d'exécuter (ou finir d'exécuter une ou plusieurs instructions) en parallèle.



## Async await

On va pouvoir indiquer à javascript sur certaines instructions d'attendre que celle ci soit complétée avant de passé à l'instruction suivante.

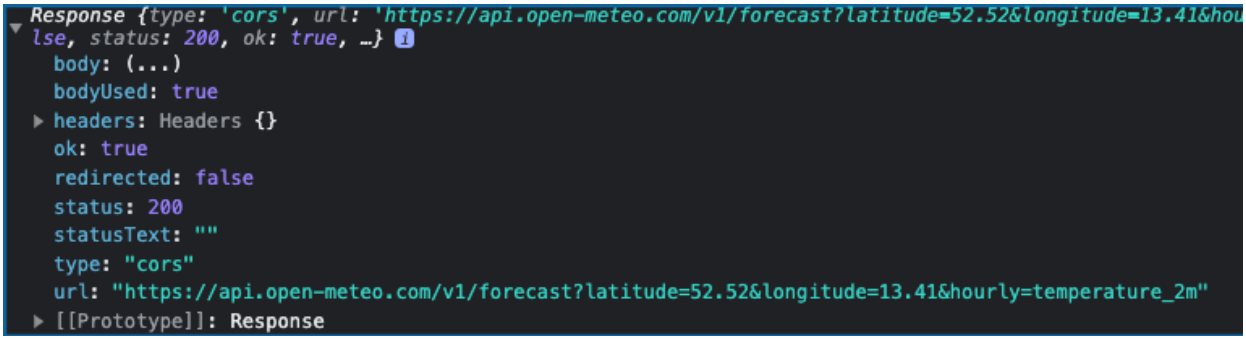
On déclare une fonction avec le mot clé **async** pour indiquer que le code contenu dans cette fonction devra être exécuté de manière asynchrone et sur certaines instructions de cette fonction utiliser le mot clé **await** (généralement sur la fonction fetch( ) et la fonction json( ) qui manipule les données de l'api)

Exemple ci dessous on contacte une url d'api météo pour afficher la latitude dans la page web :
Pour cet exemple on fait une fonction fléchée anonyme stocké dans une variable (pour s'habituer à ce genre de syntaxe), on précise avant la fonction qu'elle en mode **async** puis quand on contact l'api via **fetch**( ) on précise que cette instruction est en mode **await** elle doit se finir totalement pour continuer la suite du programme, et quand on transforme la réponse de l'api avec la fonction **json**( ), afin de transformer la réponse en objet javascript (plus facile à manipuler), idem on précise que c'est en mode **await.**

```
const apiDiv = document.querySelector('.apiContact');
//de base une f° => est anonyme, astuce pour désanonymiser, on la stocke dans une
variable
const contactApi =  async () => {
    //Data va récup Toutes les données de l'api
    const data = await fetch('https://api.open-meteo.com/v1/forecast?
latitude=52.52&longitude=13.41&hourly=temperature_2m');
    console.log(data);
    //Plutôt que de Travailler sur la réponse, on va la transformé pour
    //qu'elle devient un OBJET JS (+ pratique)
    const dataTransformed = await  data.json();
    console.log(dataTransformed);
    apiDiv.innerText = dataTransformed.latitude;
};
contactApi();
```

Prenez le réflexe de toujours se référence à la documentation officielle des API

Ci dessous le console log de data (les données brutes (la response) que renvoie l'api




C'est un objet de type **Response** qui contient plusieurs propriétés comme ok ou status : 200 (le contact avec l'api s'est bien passé)
On peut aussi gérer cela en JS pour des cas d'erreurs…

Le second console log correspond au données transformée en objet JS via la fonction .json( )
C'est là que l'on peut voir les données fournit par l'api, une fois transformée on accède aux données comme en mode objet

```
dataTransformed.latitude
```

# Exercice : Contacter une API

Avec ce EndPoint d'api
https://pokeapi.co/api/v2/pokemon

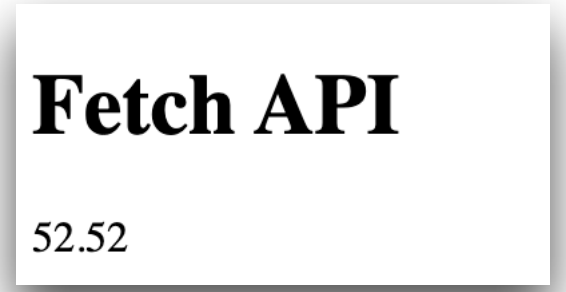
Affichez dans une page web le nom des 20 premiers Pokemon

## Then catch

Autre manière de gérer le code de manière asynchrone avec le chainage à la fonction fetch( ), des fonction(s) then( ) et enfin la fonction catch( ) pour capter et gérer les erreur

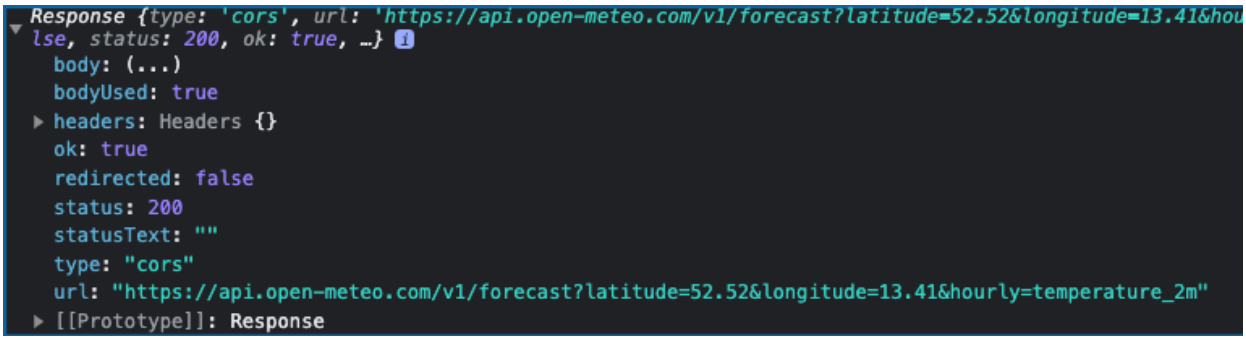
Si on adapte cette méthode à notre premier exemple d'api (celle de la météo) :

```
// //** METHODE avec Fetch + .then() + catch() */
const apiDiv = document.querySelector('.apiContact');
console.log(apiDiv);
const contactApi = () => {
    fetch('https://api.open-meteo.com/v1/forecast?
latitude=52.52&longitude=13.41&hourly=temperature_2m')
    .then(response => response.json())
    .then(data =>(apiDiv.innerText = data.latitude))
    .then(data =>(console.log(data)))
    .catch(error => console.log("Erreur custom : " + error));
};
contactApi();
```



# Gestion des erreurs Response ET Promise

Pour aller encore plus loin on peut également, en plus des erreurs de la Promise, via JS gérer les erreurs également au niveau de la response en accédant au propriétés de l'objet rappelez vous : Ci dessous le code permet de gérer dès la response une erreur si l'url du fetch est invalide.
C'est mieux pour travailler en collaboratif et assurer un aspect **qualitatif** de votre code




```
//** METHODE avec Fetch +then + catch + async Await */
const apiDiv = document.getElementById("apiContact");
const contactApi = () => {
    //! tester si jamais on se trompe dans l'url (mettre l'un des 2 fetch en
commentaire)
fetch("https://api.npms.io/on-s-est-trompe-dans-l-url")
//! Ci dessous avec une url valide
// fetch("https://api.open-meteo.com/v1/forecast?
latitude=52.52&longitude=13.41&hourly=temperature_2m")
    .then(async (response) => {
    const dataTransformed = await response.json();
    // Ici on gère aussi les erreurs au niveau de la
    // réponse de l'api
    //  Si dans la response la propriété ok n'est pas définie
    if (!response.ok) {
        // on récupère les messages d'erreur ou la propriété statusText par default
de la response
        const error = (dataTransformed && dataTransformed.message) ||
response.statusText;
        //f° native de JS utilisé sur les objets de type Promise
        return Promise.reject(error);
    }

    apiDiv.innerText = dataTransformed.latitude;
    })
    .catch((error) => {
    console.log(error);
    // Ici on crée une error custom et l'objet error
    console.error("Attention une fusée à décollée depuis Grenoble", error);
    });
};
contactApi();
```