

JAVASCRIPT

Durée Totale du Module : 21H

Auteur :

Jean-François Pech

Date création :

03/03/2023

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Gestion des erreurs

Try ... Catch

Aspect très important de la programmation d'autant plus quand on travaille dans une équipe, progressivement vos applications vont se complexifier et contenir beaucoup de code, d'objet, de fonctions etc..

Il y a des erreurs plus ou moins grave et de sources différentes

Les classiques :

- Erreur de Syntaxe, oubli d'un « ; », length ou lenght ?, etc...
- Erreur qui vient du serveur (pratique on peut accuser les développeurs BackEnd, l'incapacité à charger un fichier (404, etc...))
- Erreur qui vient du navigateur
- Erreur qui vient de l'utilisateur (envoyer une valeur non conforme dans un formulaire par exemple)

Dans la majorité des cas, une erreur va provoquer l'arrêt brutal d'un script et on risque donc d'avoir des codes et des pages non fonctionnelles.

Dans le pire des cas, une erreur peut être utilisée comme faille de sécurité par des utilisateurs malveillants qui vont l'utiliser pour dérober des informations ou pour tromper les autres visiteurs.

Javascript comporte nativement des fonctionnalités pour gérer les cas d'erreurs.

Dans tous les cas vous avez pu le constater, quand votre code comporte une erreur (vous avez un message d'erreur de base dans la console du navigateur), cela signifie que de base Javascript va créer, générer une erreur à partir de l'objet global Error (un objet de base dans javascript)

Par la suite nous allons pouvoir capturer l'objet d'Error renvoyé par Javascript et pouvoir indiquer ce que l'on souhaite faire dans le code quand cette erreur survient.

On va avoir à disposition un syntaxe de bloc try ... catch...

Dans le bloc try : on écrit le code à tester (qui peut potentiellement générer des erreurs)

Dans le bloc catch : on écrit le code pour gérer l'erreur (on fait un simple console.log ? On affiche un message à l'utilisateur ? On enregistre quelque chose en base de données ?

Nous allons voir le gestion d'erreurs au travers d'exemples.

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

```
prenom;  
alert('Ce message ne s\'affichera pas');
```

Erreur générée par la fonction alerte

✖ Uncaught SyntaxError: Invalid or unexpected token (at app.js:4:7) app.js:4

Erreur générée par la variable prénom

✖ ▶ Uncaught ReferenceError: prenom is not defined app.js:3
 at app.js:3:1

Donc si on sait que ce code d'exemple peut nous créer des erreurs on peut l'écrire de cette manière :

```
try{  
  prenom  
  alert('Bonjour');  
}catch(err){  
  alert(`Erreur Détectée ALERTE STOPPEZ TOUT :  
    -----  
    Le Nom de l'erreur  
    ${err.name}  
    -----  
    Le Message de l'erreur :  
    ${err.message}  
    -----  
    L'emplacement de l'erreur:  
    ${err.stack}`);  
}  
alert(`Ce message s'affiche correctement`);
```

Comme on a un erreur dans le code du bloc Try seule les alertes du bloc catch puis ensuite la dernière alert sera affichée à l'utilisateur.

Auteur :

Jean-François Pech

Relu, validé & visé par :

☑ Jérôme CHRETIENNE
 ☑ Sophie POULAKOS
 ☑ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.



127.0.0.1:5500 says

Erreur Détectée ALERTE STOPPEZ TOUT:

Le Nom de l'erreur
ReferenceError

Le Message de l'erreur :
prenom is not defined

L'emplacement de l'erreur:
ReferenceError: prenom is not defined
at http://127.0.0.1:5500/app.js:7:5

☐ Don't allow this page to display more dialogs

OK



127.0.0.1:5500 says

Ce message s'affiche correctement

☐ Don't allow this page to display more dialogs

OK

Auteur :

Jean-François Pech

Date création :

03/03/2023

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Gestion des exceptions avec Throw

Dans certains cas, lorsque l'on écrit les différentes fonctions d'un programme, il se peut que le code soit juste mais cela ne correspond plus à la réalité.

Imaginez vous réalisez une application pour gérer des retrait et virement bancaires, vous avez 10 euros sur votre compte et vous voulez faire un retrait de 50 K€, techniquement si on se place du point de vue du code il nous faudra certainement une fonction qui fait une soustraction.

Donc du point de vue du code on peut faire 10-50000, ça va fonctionner et votre solde sera de -49990 €.

Le code est juste d'un point de vue technique mais du point de vue du Banquier peut être pas.

Dans notre cas il faut bien prendre en compte les règles de gestion de l'application et du corps de métier, et donc ici il nous faudra mettre en place une exception dans le cas où le montant que voudrait retiré serait supérieur au solde de l'utilisateur.

Auteur :

Jean-François Pech

Date création :

03/03/2023

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Ici ce programme a pour but de demander 2 données à l'utilisateur (la fonction prompt), pour ensuite diviser ces données.

On va donc mettre en place 2 exceptions (il pourrait y en avoir plus)

1 exception si l'utilisateur ne renseigne pas 2 nombres.

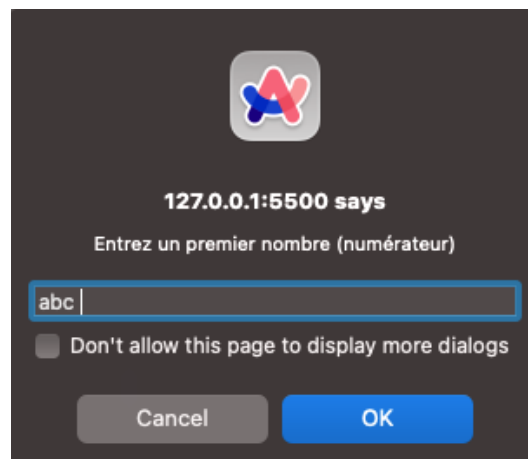
1 exception si l'utilisateur essaye de nous faire diviser par zéro

```
function division(){
  let x = prompt('Entrez un premier nombre (numérateur)');
  let y = prompt('Entrez un deuxième nombre (dénominateur)');

  if(isNaN(x) || isNaN(y) || x == '' || y == ''){
    throw new Error('Merci de rentrer deux nombres');
  }else if(y == 0){
    throw new Error('Division par 0 impossible')
  }else{
    alert(x / y);
  }
}

division();
```

1 ère exception :



✖ ▶ Uncaught Error: Merci de rentrer deux nombres
 at div (app.js:29:15)
 at app.js:37:1

app.js:29

Auteur :

Jean-François Pech

Relu, validé & visé par :

☑ Jérôme CHRETIENNE
 ☑ Sophie POULAKOS
 ☑ Mathieu PARIS

Date création :

03/03/2023


Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

2^e exception :



127.0.0.1:5500 says

Entrez un deuxième nombre (dénominateur)

☐ Don't allow this page to display more dialogs

Cancel OK

✖ ▶ Uncaught Error: Division par 0 impossible
 at div (app.js:31:15)
 at app.js:37:1

app.js:31

Auteur :

Jean-François Pech

Date création :

03/03/2023

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Finally

Au sein d'un bloc try catch, on peut (optionnel) utiliser l'instruction Finally, ce bloc nous permet de préciser du code qui sera exécuté dans tous les cas, qu'une erreur ou exception ait été générée ou pas.

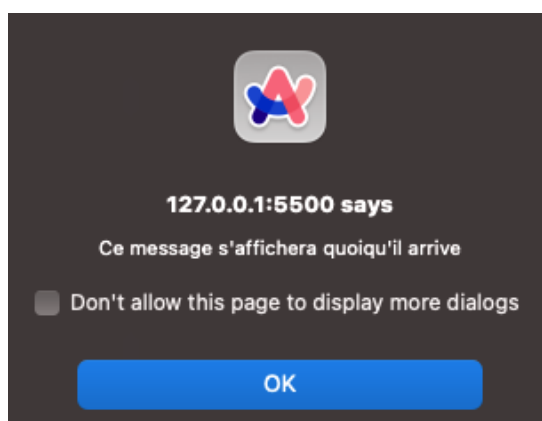
Reprenons notre fonction division : ici on ne va plus exécuter la fonction division directement dans notre programme on va **essayer** d'exécuter la fonction.

```
function division(){
    let x = prompt('Entrez un premier nombre (numérateur)');
    let y = prompt('Entrez un deuxième nombre (dénominateur)');

    if(isNaN(x) || isNaN(y) || x == '' || y == ''){
        throw new Error('Merci de rentrer deux nombres');
    }else if(y == 0){
        throw new Error('Division par 0 impossible');
    }else{
        alert(x / y);
    }
}

// division();

try{
    division();
}catch(err){
    alert(err.message);
}finally{
    alert(`Ce message s'affichera quoiqu'il arrive`);
}
```



Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Les Classes

Pré requis : Les objets

Les classes sont un élément essentiel de le P.O.O (Programmation orientée objet), pour résumer le concept les classes vont nous permettre de créer plus facilement et rapidement plusieurs objets avec des caractéristiques, d'architectures similaires.

Par exemple si une application gère des utilisateurs on peut définir une classe « user » et pour chaque user on gère les données suivantes : un nom, un mail, un num de téléphone.

Pour créer / **construire** des nouveaux user, le système de class utilise une fonction qui s'appelle **constructor** (**construct** dans d'autres langages)

On va pouvoir plus facilement créer de nouveaux user (des nouvelles instances de la classe user)

Classe

User	
Nom	<input type="text"/>
Mail	<input type="text"/>
Tél	<input type="text"/>

Instance

User	
Nom	José
Mail	Jose@gmail.com
Tél	098765373

Instance

User	
Nom	SarahConor
Mail	sarah@gmail.com
Tél	12345678909

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Syntaxe

Donc côté code, on crée notre class UserProfile, pour pouvoir créer des nouvelles instance on renseigne les données dont on a besoin, que l'on recevra en paramètre (nameUser, mailUser, phoneUser)

Le mot clé this va représenter l'objet courant, celui que l'on est entrain de créer, c'est le contexte. Ici pour résumer on assigne aux propriété de notre classe les valeur qu'on va recevoir en paramètre

```
class UserProfile {  
    constructor(nameUser, mailUser, phoneUser) {  
        this.nameUser = nameUser;  
        this.mailUser = mailUser;  
        this.phoneUser = phoneUser;  
    }  
    getProfileInfo() {  
        return `infos de l'utilisateur :  
            son nom : ${this.nameUser}  
            son mail : ${this.mailUser}  
            son Tél : ${this.phoneUser}`;  
    }  
}
```

Bonus : on a écrit une fonction au sein de la classe, c'est une méthode de classe et elle ne pourra s'utiliser QUE sur des objets (des nouvelles instances) de cette classe

Une fois qu'on a défini la structure de notre classe on va pouvoir utiliser le constructeur pour créer un nouvel utilisateur en faisant new UserProfile()

```
const exampleUser1 = new UserProfile("José", "jose@gmail.com",  
« 09876543»);
```

```
const exampleUser2 = new UserProfile("Sarah", "sarah@gmail.com",  
"063736252");  
exampleUser2.getProfileInfo();
```

Dans notre cas il n'y aura que sur des new UserProfile que l'on pourra utiliser la méthode getProfileInfo().

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

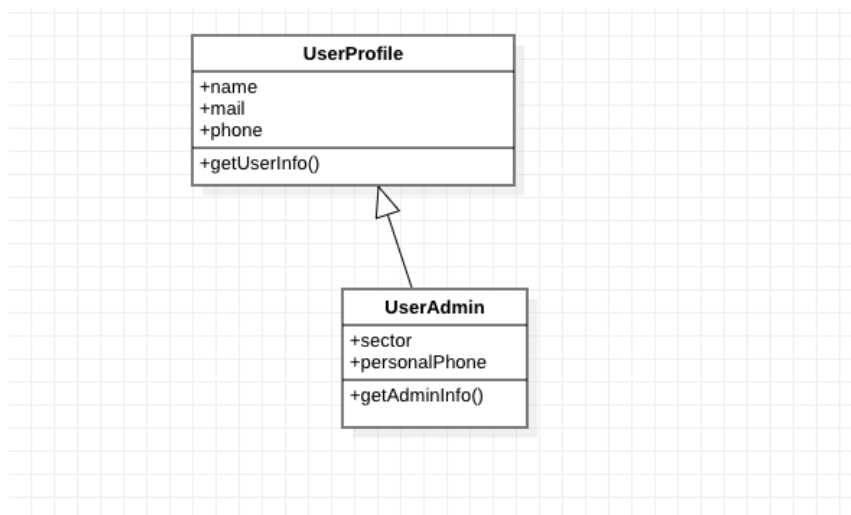
10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

L'héritage

Avec les classes on peut également profiter d'un système d'héritage, cela signifie que nous pouvons étendre (**extends**) les propriétés, les méthodes d'une classe vers une autre, Par exemple dans notre application on gère déjà des utilisateurs, mais on veut aussi gérer des utilisateurs un peu plus spécifiques : des Admin, les admins ils auraient les mêmes propriétés que les utilisateurs (un nom, un mail, un téléphone) mais avec des informations en plus (le **secteur** dans lequel l'admin travaille, et son **Téléphone personnel**)
 on crée une nouvelle classe « enfant » qui hérite des propriétés et des méthodes d'une classe parent.



⚠ une classe enfant peut hériter d'une classe parent mais l'inverse n'est pas possible.
 Sur une instance de **UserAdmin** on pourra utiliser **getProfileInfo()** mais sur une instance de **UserProfile** on ne peut pas utiliser **getAdminInfo()**.

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Dans le code : on va utiliser **extends** et **super()**

```
class UserProfile {
  //! Pas besoin de déclarer fonction devant le constructor et méthodes
  constructor(nameUser, mailUser, phoneUser) {
    this.nameUser = nameUser;
    this.mailUser = mailUser;
    this.phoneUser = phoneUser;
  }
  getProfileInfo() {
    return `infos de l'utilisateur :
      son nom : ${this.nameUser}
      son mail : ${this.mailUser}
      son Tél : ${this.phoneUser}`;
  }
}

const exampleUser1 = new UserProfile("José", "jose@gmail.com", "09876543");
const exampleUser2 = new UserProfile("Sarah", "sarah@gmail.com", "063736252");
exampleUser2.getProfileInfo();

class UserAdmin extends UserProfile{
  constructor(unNom,unMail,unPhone,sector,personnalPhone){
    super(unNom,unMail,unPhone); //! Appel au constructor du parent
    this.sector = sector;
    this.personnalPhone = personnalPhone;
  }
  getAdminInfo(){
    return `infos de l'utilisateur :
      son nom : ${this.nameUser}
      son secteur d'intervention : ${this.sector}
      son Tél Personnel : ${this.personnalPhone}`;
  }
}

const exampleAdmin1 = new UserAdmin('Jacky','jack@gmail.com','012345678','administration','0987654323');
console.log(exampleAdmin1.getAdminInfo());
```

Auteur :

Jean-François Pech

Relu, validé & visé par :

☑ Jérôme CHRETIENNE
 ☑ Sophie POULAKOS
 ☑ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Exo Class IMC

Créer un programme permettant de Calculer l'IMC d'une personne

TODO :

- Créer une classe Imc avec un constructeur qui recevra un nom, un poids, une taille
- Créer une fonction de calcul d'IMC, qui retourne le résultat du calcul (à 2 nombre après la virgule si possible)
- Créer une fonction d'affichage « display », elle a pour rôle d'afficher en console :
Le nom de la personne, son poids, sa taille et son imc dans une phrase
- En dehors de la class (donc dans le programme principal), récupérer la variable list (un tableau de nouvelle instances de la class) (voir discord ou 📌)
- Trouver un moyen de parcourir les éléments dans la variable list, sur chaque element utiliser la fonction display

En console du navigateur :

```
Sébastien Chabal (135 kg, 1.7 M) a un IMC de: 46.71
Escaladeuse (45 kg, 1.68 M) a un IMC de: 15.94
JOJO (300 kg, 2 M) a un IMC de: 75.00
Gontrand (90 kg, 1.75 M) a un IMC de: 29.39
Colonel Clock (200 kg, 1.75 M) a un IMC de: 65.31
J0siane de la Vega (99 kg, 1.55 M) a un IMC de: 41.21
```

Programme Principal (en dehors de la classe)

```
// /** progr principal -> on fait l'injection des données
let list = [
  new Imc("Sébastien Chabal", 135, 1.7),
  new Imc("Escaladeuse", 45, 1.68),
  new Imc("JOJO ", 300, 2),
  new Imc("Gontrand ", 90, 1.75),
  new Imc("Colonel Clock ", 200, 1.75),
  new Imc("J0siane de la Vega", 99, 1.55),
];
/**Boucle qui parcourt list pour utiliser display()
?????.????????((????????) ?? ????.????());
```

Auteur :

Jean-François Pech

Relu, validé & visé par :

☑ Jérôme CHRETIENNE
 ☑ Sophie POULAKOS
 ☑ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

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 devons 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 renvoient plusieurs types de réponse.

Actuellement la plupart des API sont des API **restful** c'est-à-dire que le format des réponses que renvoie l'API peut être en **JSON**, HTML, XML, 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.

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Fetch()

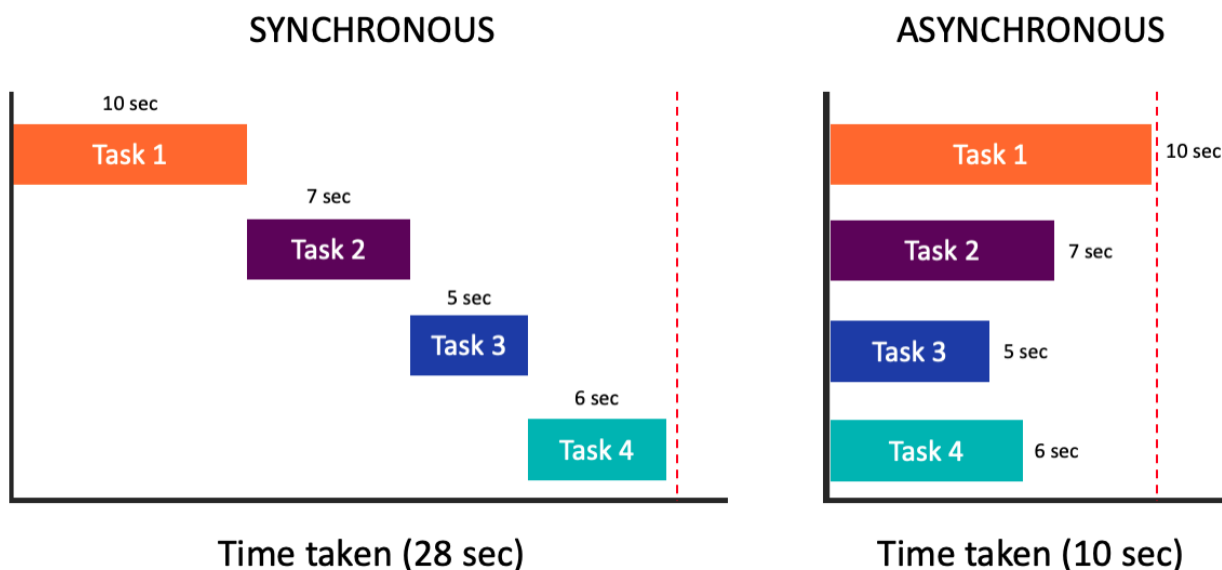
Dans Javascript, nous allons utiliser la méthode `fetch()`, qui nous permettra de contacter n'importe quelle API via son URL, la méthode renvoi des objets de type **Response** ou **Promise**.
 L'API Fetch et sa méthode `fetch()` qui correspondent à la "nouvelle façon" d'effectuer des requêtes HTTP.

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.



Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Async await

On va pouvoir indiquer à javascript sur certaines instructions d'attendre que celle ci soit complétée avant de passer 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 stockée dans une variable (pour s'habituer à ce genre de syntaxe), on précise avant la fonction qu'elle est en mode **async** puis quand on contacte 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 récupérer 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 transformer pour qu'elle devienne un OBJET JS (+ pratique)  
  const dataTransformed = await data.json();  
  console.log(dataTransformed);  
  apiDiv.innerText = dataTransformed.latitude;  
};  
contactApi();
```

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

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

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

```
Response {type: 'cors', url: 'https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&hourly=temperature_2m', status: 200, ok: true, ...}
  body: (...)
  bodyUsed: true
  headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&hourly=temperature_2m"
  [[Prototype]]: Response
```

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`

```
{latitude: 52.52, longitude: 13.419998, generationtime_ms: 0.35691261291503906, utc_offset_second
s: 0, timezone: 'GMT', ...}
  elevation: 38
  generationtime_ms: 0.35691261291503906
  hourly: {time: Array(168), temperature_2m: Array(168)}
  hourly_units: {time: 'iso8601', temperature_2m: '°C'}
  latitude: 52.52
  longitude: 13.419998
  timezone: "GMT"
  timezone_abbreviation: "GMT"
  utc_offset_seconds: 0
  [[Prototype]]: Object
```

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

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

bulbasaur

ivysaur

venusaur

charmander

charmeleon

charizard

squirtle

wartortle

blastoise

caterpie

metapod

butterfree

weedle

kakuna

beedrill

pidgey

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

Then catch

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

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();
```

Fetch API

52.52

Auteur :

Jean-François Pech

Relu, validé & visé par :

☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

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 réponse en accédant aux propriétés de l'objet rappelez vous :
Ci dessous le code permet de gérer dès la réponse une erreur si l'url du fetch est invalide.
C'est mieux pour travailler en collaboratif et assurer un aspect **qualitatif** de votre code

```
▼ Response {type: 'cors', url: 'https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&hourly=temperature_2m'}
  lse, status: 200, ok: true, ...
  body: (...)
  bodyUsed: true
  ▶ headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitude=13.41&hourly=temperature_2m"
  ▶ [[Prototype]]: Response
```

```
/** 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 réponse la propriété ok n'est pas définie
    if (!response.ok) {
      // on récupère les messages d'erreur ou la propriété statusText par default
      de la réponse
      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();
```

Auteur :

Jean-François Pech

Relu, validé & visé par :

☑ Jérôme CHRETIENNE
☑ Sophie POULAKOS
☑ Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023



Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.