

# 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.

## Les Classes

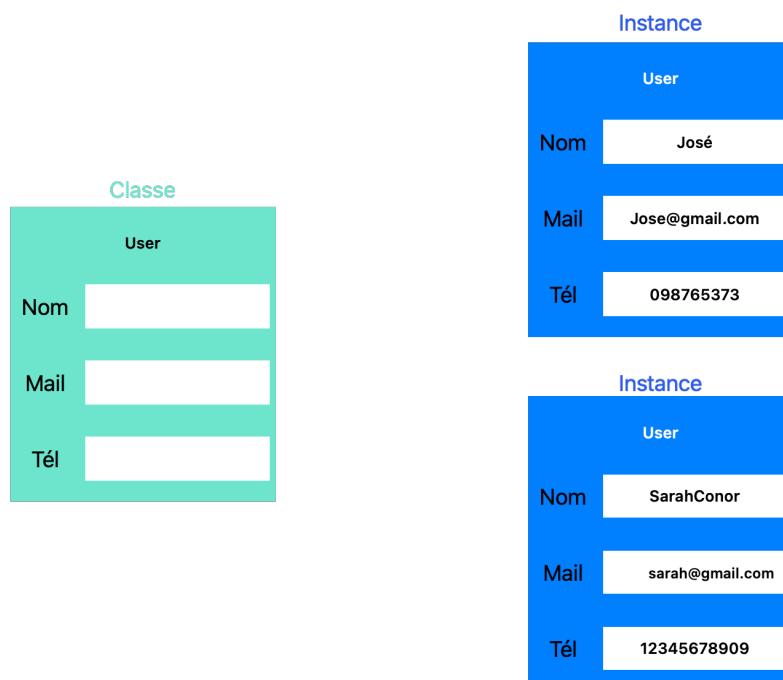
Pré requis : Les objets

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)



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.

## 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 définit 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().

## 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

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.

## Exo Class PME

Gérer une PME

CDC

Un Salarié a un nom, prénom, âge, salaire mensuel  
Il est payé sur N mois.  
En plus il y a XXX de charges

Une Pme c'est un nom, une équipe de plusieurs salariés  
Grâce à ses ventes elle a des revenus R  
Mais aussi ... :  

- des frais fixes FF (impôts etc...)
- Des frais d'achats de matériel et de logiciels FA

TODO :

- Créer une classe Pme et une classe Employee
- Utiliser des fonctions
- Faire le bilan annuel de l'entreprise et l'afficher en console.

Détails :

- 3 salariés qui gagnent par mois : 2000, 3000 et 4000 euros
- R = 300000 (trois cent mille)
- FF = 20000 (vingt mille)
- FA = 50000 (cinquante mille)
- N = 12
- XXX = 90%

En console du navigateur :

```
-----MA PME-----
Ma Petite Entreprise - : Cout Initial : 70000
Ma Petite Entreprise - : Cout Total Equipe : 205200
Ma Petite Entreprise - : VENTES : 300000
Ma Petite Entreprise - : BILAN : 24800
```

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.

```
// // Scénario
const pme = new Pme (
    //Le nom entreprise
    "Ma Petite Entreprise -",
    //L'équipe de salariés (un tableau)
    [new Employee ("Duval", "Paul", 30, 2000),
     new Employee ("Durand", "Alain", 40, 3000),
     new Employee ("Dois", "Sylvia", 50, 4000)],
    //le revenu , frais fixe, frais d'achat
    300000,
    20000,
    50000);
pme.bilanCalculed();
```

**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 COMPTES BANCAIRES.

### Enoncé

Gérer des compte en banque

### Consignes

- Créer une classe CompteBancaire avec des méthodes de crédit, de retrait, de visualisation de l'état du compte bancaire (en console), on doit pouvoir aussi faire un virement d'un membre à un autre.
- Générer une exception pour ne pas dépasser le solde (pas de retrait ou de virement qui dépassent le solde du compte bancaire)

### Détails

Faire une scénario avec gestion de 3 comptes crédités à 1000 € chacun (Alex, Clovis, Marco)  
Puis Alex retire 100

Puis Marco fait un virement de 300 à Clovis

Enfin Alex tente un retrait de 1200

Afficher tous les soldes finaux.

Ces compte sont placés dans un tableau associatif de clients

En console :

```
Ajout de: 1000 pour: Alex
Ajout de: 1000 pour: Clovis
Ajout de: 1000 pour: Marco
Retrait de: 100 pour: Alex
Virement de: 300 de: Marco vers: Clovis
Ajout de: 300 pour: Clovis
Retrait de: 300 pour: Marco
----->Alex, retrait de: 1200 refusé avec solde: 900
titulaire: Alex, solde: 900
titulaire: Clovis, solde: 1300
titulaire: Marco, solde: 700
```

Des ressources :

[https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Working\\_with\\_objects](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Working_with_objects)

<https://www.pierre-giraud.com/javascript-apprendre-coder-cours/gestion-erreur-exception-try-catch/>

<https://www.pierre-giraud.com/javascript-apprendre-coder-cours/gestion-erreur-exception-try-catch/>

<https://javascript.info/try-catch>

#### 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.

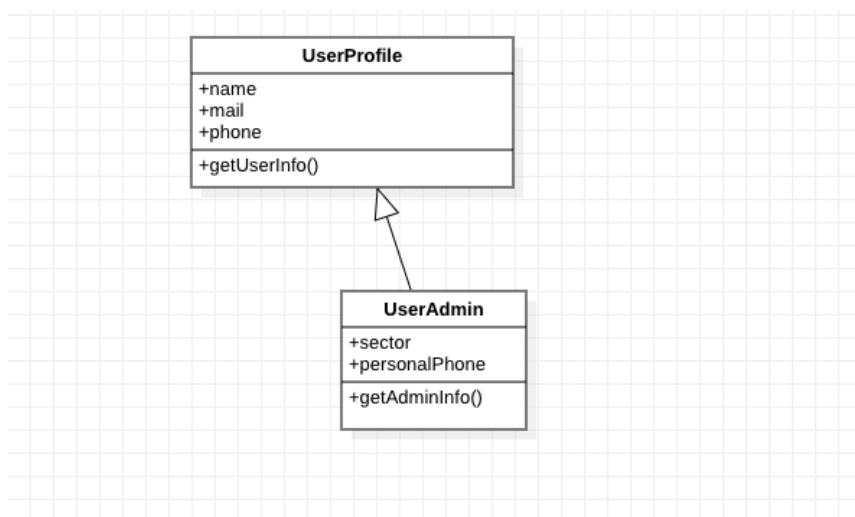
```
// Main, gère 3 comptes bancaires dans un tableau associatif
const lesComptes = {
  Alex: new CompteBancaire("Alex"),
  Clovis: new CompteBancaire("Clovis"),
  Marco: new CompteBancaire("Marco"),
};

// lecture tableau associatif ou Objet["truc"]
// Crédite et décrète chaque compte
for (let key in lesComptes) lesComptes[key].crediter(1000);

// un retrait de 100 par Alex
??????????????;
// un petit virement: Marco Vire 300 à Clovis
??????????????;
// un petit retrait incorrect (doit déclencher erreur custom) :
// Alex fait un retrait de 1200
?????;
// bilan : faire une description de tous les comptes en console (ou DOM ?)
?????;
```

## 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

### 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.

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

```

class UserProfile {
    //! Pas besoin de déclarer function 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());

```

Penser à rajouter exemple avec attributs private

**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.

## La guerre des langages

En programmation on peut distinguer à peu près tous les langages en 2 familles, les langages basés sur les **classes** et ceux basés sur les **prototypes**.  
 Js est un langage orienté objet basé sur les prototypes. (C'est pour cela que l'on dit qu'en Javascript TOUT est Objet)

Le JavaScript est un langage objet basé sur les prototypes. Cela signifie que le JavaScript ne possède qu'un type d'élément : **les objets** et que tout objet va pouvoir partager ses propriétés avec un autre, c'est-à-dire servir de prototype pour de nouveaux objets.  
 L'héritage en JavaScript se fait en remontant la chaîne de prototypage.

En plus de la manière déclarative (créer une variable et lui assigner directement une valeur) dans Javascript on va retrouver également un système de constructor pour créer tout type d'objet

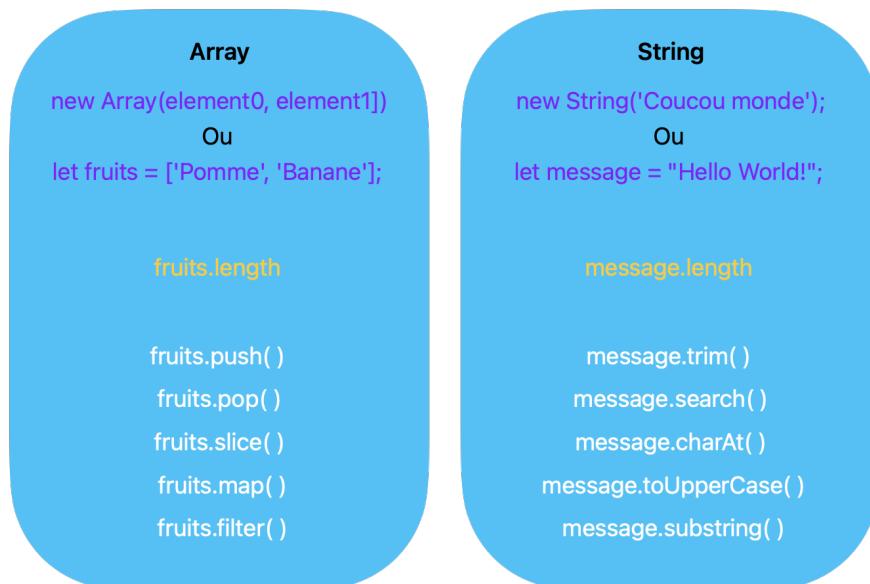
Exemple de fiche récapitulative des types d'objets en JS (non exhaustif) :

On peut déclarer soit en utilisant le constructor `new Array()`, mais il faut penser à stocker dans une variable, nativement dans JS pour chaque objet on aura des propriétés de base, ici `length` commun à plusieurs type et JS propose aussi des fonctions de base, utiles pour manipuler chaque type d'objets. (Toujours avoir le réflexe d'aller consulter la documentation, NE PAS réinventer la ROUE)

Violet : créer une variable (via constructor ou en mode déclaratif)

Jaune : exemple d'une propriété

Blanc : des fonctions de bases



### 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.

## 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.

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.

## Fetch()

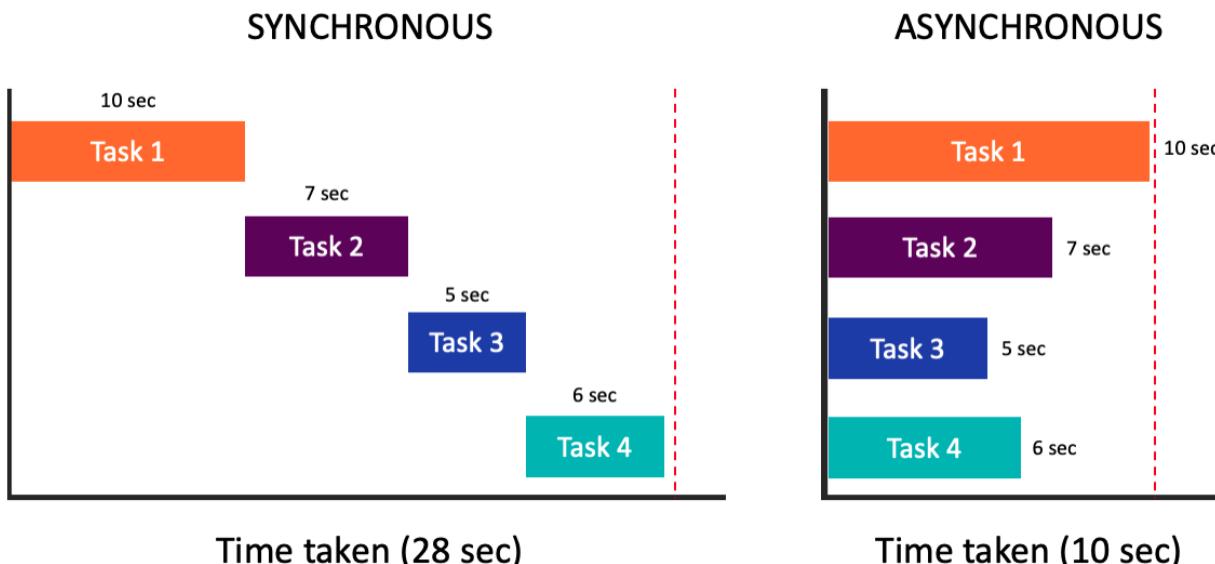
Dans Javascript, nous allons utiliser la méthode `fetch()`, qui nous permettra de contacter n'importe quelle API via son URL, la méthode renvoi des objets de type **Response** ou **Promise**.  
l'API Fetch et sa méthode `fetch()` qui correspondent à la “nouvelle façon” d’effectuer des requêtes HTTP.

## 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

### 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.

## 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 stocké 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 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 référer à la documentation officielle des API

### 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.

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

```
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: (...)

  ▶ 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`

```
app.js:40
  ▼ {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



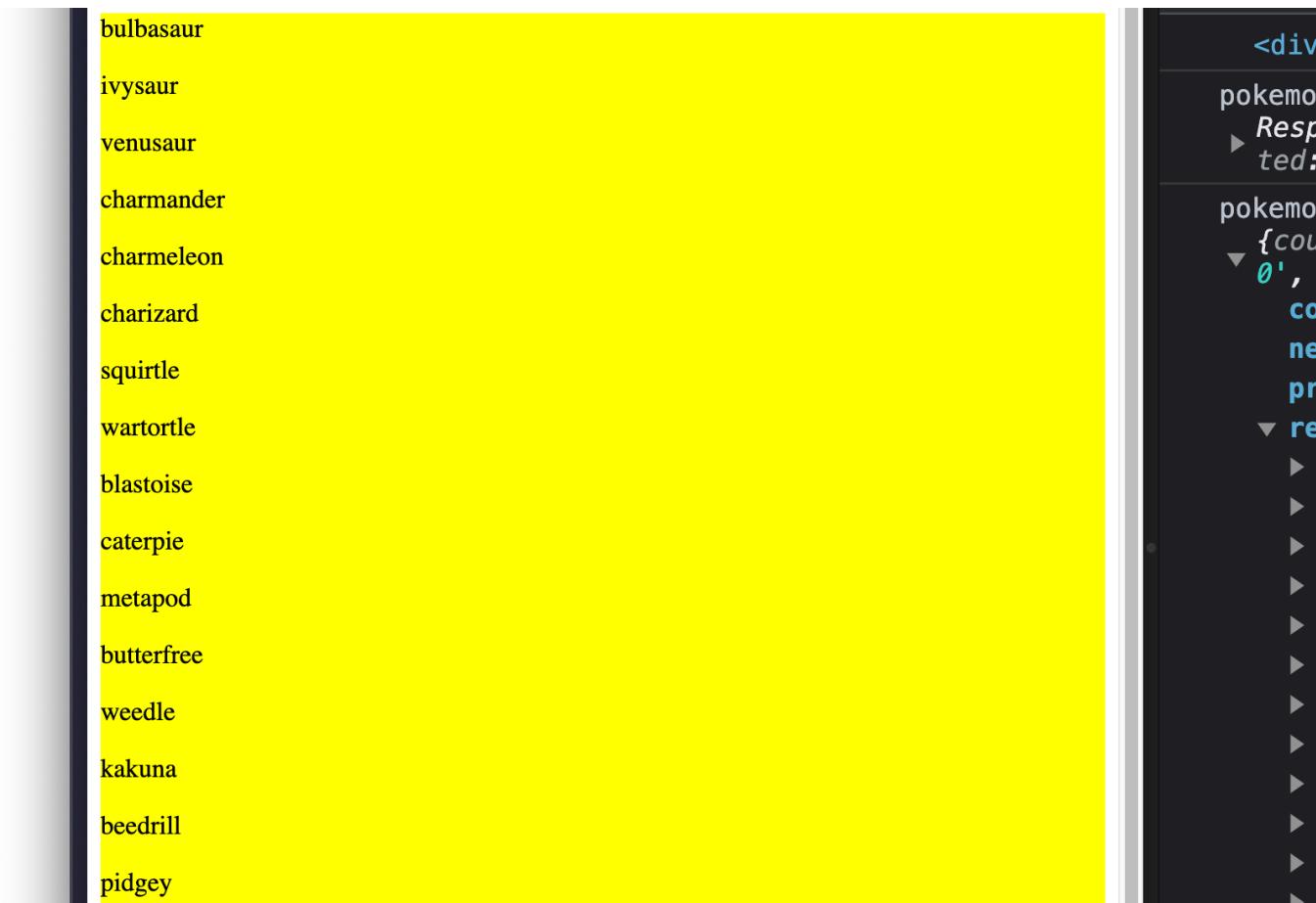
Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

## 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



## 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 erreurs

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

```
// //** MÉTHODE 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

#### 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 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

```
▼ 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
```

```
/** 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
                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.

## Javascript Modulaire

Le JavaScript modulaire est une approche qui permet de structurer le code en le divisant en modules réutilisables et indépendants.

Cette méthode facilite la gestion du code, améliore la lisibilité et permet de réduire les conflits de noms.

Grâce aux fonctionnalités d'importation et d'exportation, les développeurs peuvent partager des fonctions, des objets ou des classes entre différents fichiers, ce qui favorise la réutilisation du code et la collaboration au sein des équipes.

En utilisant des modules, il devient également plus simple de tester et de maintenir le code, car chaque module peut être développé et testé de manière isolée.

Cette approche est devenue essentielle dans le développement moderne, notamment avec l'émergence de frameworks et de bibliothèques qui exploitent pleinement les capacités des modules JavaScript.

Au préalable on peut rajouter sur la balise script de notre dans notre HTML avec l'attribut

Type « module »

```
<script type="module" src="/main.js"></script>
```

Exemple avec un premier fichier hello.js qui va exporter une fonction

```
export function moduleHello(name) {
    console.log('Bonjour depuis hello.js ' + name);
    // alert(`Bonjour ${name}`);
    // return `Bonjour depuis hello.js : ${name}`;
}
```

Ensuite dans un autre fichier JS nous allons pouvoir importer et utiliser cette fonction.

```
import { moduleHello } from './scripts/hello';

moduleHello('jojo');
```

**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.

## Gérer les évènements du DOM

Pour pouvoir prétendre à des application totalement « réactive » il va nous falloir cette dernière étapes à savoir, le fait de pouvoir réagir lorsque l'utilisateur déclenche un évènement dans la page. (Le click sur un bouton, le fait de scroller ou encore envoyer un formulaire).

La fonction **addEventListener** qui prend 2 arguments en paramètres, une chaîne de caractère pour mentionner quel évènement on doit surveiller (click, scroll, etc...) et en second paramètre, une fonction qui va réunir plusieurs instructions.

```
//? Mode f° => (anonyme + fléchée)
let leH1 = document.querySelector('#mainTitle');

leH1.addEventListener('click', ()=>{
    console.log('ok ca click');
});
```

Dans l'exemple ci-dessus on sélectionne un titre h1 via son id, ensuite sur cet élément HTML on écoute le click et en réaction on fait un affichage en console. (Ici on utilise une fonction fléchée en second paramètre)

Ci-dessus avec une syntaxe classique de fonction mais également anonyme (généralement on

```
//? Mode fonction anonyme classique
leH1.addEventListener('click', function(){
    console.log('ok ca click');
});
```

utilise des fonctions anonymes dans les addEventListener, on a pas vraiment besoin de rappeler ces fonctions dans d'autres parties du programme), on peut aussi exécuter une fonction déclarée ailleurs dans le programme comme ci-dessous 

```
// ? la fonction est en dehors
function reactClick(){
    console.log('ok ca click');
}

leH1.addEventListener('click', reactClick);
```

Auteur :

Jean-François Pech

Relu, validé & visé par :

Jérôme CHRETIENNE  
 Sophie POULAKOS  
 Mathieu PARIS

Date création :

03/03/2023

Date révision :

10/03/2023

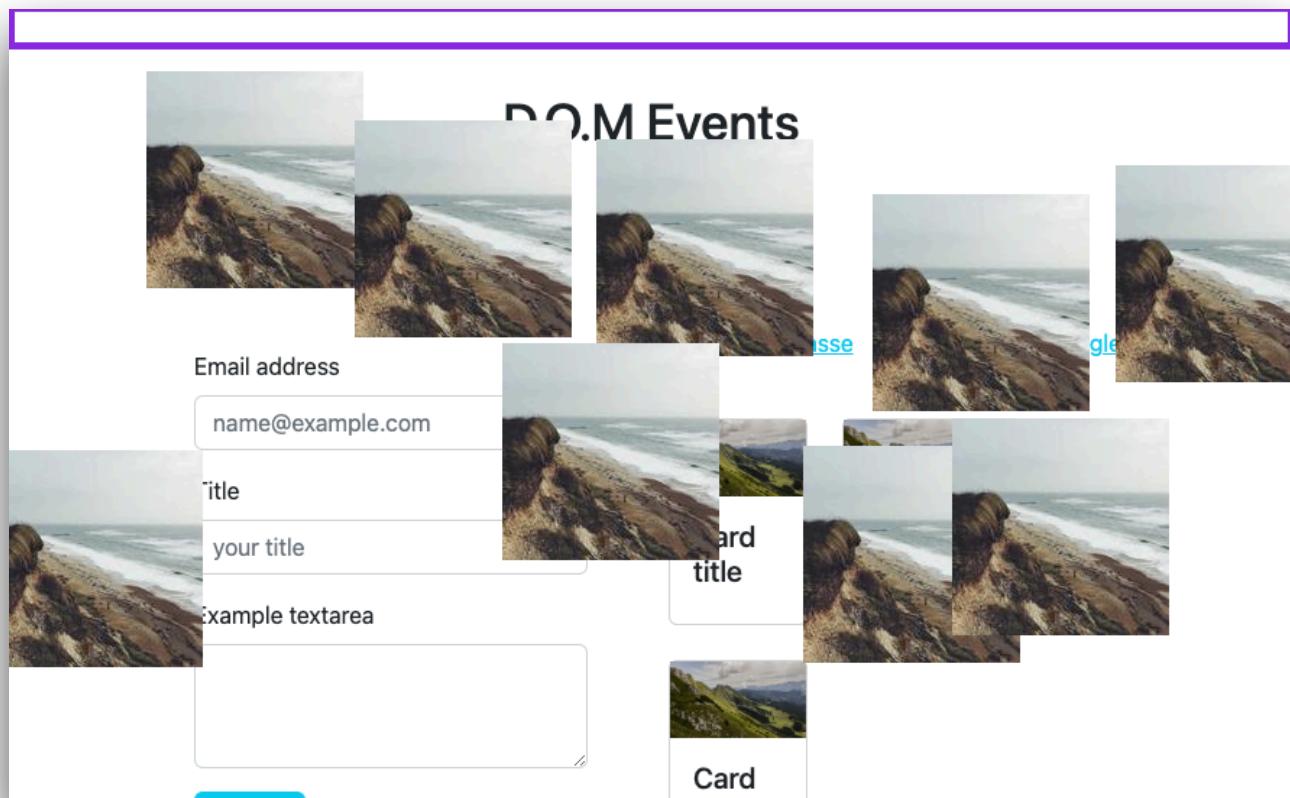


Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.

## Exo : réagir au click 3 (capter l'event)

Dans une page web faire en sorte de récupérer les coordonnées X et Y du click et créer une image à cet endroit là.

Pensez à mettre un paramètre dans la fonction du addEventListener afin de l'afficher en console (pour capter l'évènement)

**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.

## Exercice : réagir à mouseleave

Lorsque la souris de l'utilisateur s'en va d'un élément html.

Dans une page web rajouter un titre (h1, h2 ou h3 au choix) il est en display none en CSS.

Ensuite sur toute la page surveiller l'évènement mouseleave de manière à faire apparaître le titre en mettant son display en « block » (ajouter d'autre modifications du titre via js notamment au niveau du style)

Dès que le curseur de la souris quitte la page :



**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.

## Exercice DOM SetTimeout

Aller se renseigner sur l'utilisation de la fonction setT

Dans une page web, en utilisant la fonction setTimeout, faire en sorte qu'au bout de 3 secondes (soit 3000 ms), un titre apparaît sur la page et des modifications du style sont effectuées.

**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.

La page quand elle est chargée :

i http://127.0.0.1:5500

## D.O.M Events

Email address

Title

Example textarea

**Submit**

Ajouter Classe Suppr Classe Toggle Classe



Card title



Card title



Card title



Card title

1 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem quas numquam eligendi, sit obcaecati ullam dignissimos quia suscipit?

2 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem quas numquam eligendi, sit obcaecati ullam dignissimos quia suscipit?

3 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem

**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.

La page au bout de 3 secondes :

i http://127.0.0.1:5500

## D.O.M Events

### Welcome 2 the DOM

Email address

Title

Example textarea

[Ajouter Classe](#) [Suppr Classe](#) [Toggle Classe](#)



Card title



Card title



Card title



Card title

Submit

1 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem quas numquam eligendi, sit obcaecati ullam dignissimos quia suscipit?

2 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem quas numquam eligendi, sit obcaecati ullam dignissimos quia suscipit?

3 - Lorem ipsum dolor sit amet consectetur adipisicing elit. Explicabo deserunt magnam voluptas placeat, deleniti sed libero. Explicabo, officia dolores assumenda, exercitationem

**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.

## Exercice DOM + Regexp

Les regex ou expression régulière sont un syntaxe particulière pour pourvoir majoritairement tester des chaînes de caractères (on les utilise souvent dans le cadre des formulaires de connexion ou d'inscription, en plus des contrôles côté HTML), par exemple pour contrôler si l'utilisateur a bien un mot de passe sécurisé ou si un mail est bien valide par exemple.

Exemple d'une syntaxe de regex :

```
const regex = /^[a-z0-9._-]+@[a-z0-9._-]+\.\w{2,6}$/;
```

Si on analyse plus précisément :

**^** : Cela signifie que la correspondance doit commencer au début de la chaîne.

**[a-z0-9.\_-]+** : Cette partie signifie qu'au moins un caractère parmi les lettres minuscules (a à z), les chiffres (0 à 9), ou les caractères spéciaux '.', '\_', ou '-' doit être présent.

**Le '+'** indique qu'il doit y avoir au moins un caractère présent, mais il peut y en avoir plus.

**@** : C'est simplement le caractère '@', qui est obligatoire dans une adresse e-mail.

**[a-z0-9.\_-]+** : De manière similaire à la première partie, cela signifie qu'au moins un caractère parmi les lettres minuscules (a à z), les chiffres (0 à 9), ou les caractères spéciaux '.', '\_', ou '-' doit être présent avant le '@'.

**\.** : Ceci correspond à un point ('.') dans l'adresse e-mail. Notez que le point doit être échappé avec un backslash ('\') car dans les expressions régulières, un point est un métacaractère qui correspond à n'importe quel caractère.

**\w{2,6}** : Cette partie spécifie le domaine de l'adresse e-mail. Elle doit contenir uniquement des lettres minuscules (a à z), et elle doit avoir une longueur comprise entre 2 et 6 caractères.

**\$** : Cela signifie que la correspondance doit se terminer à la fin de la chaîne.

Les expressions régulières comportent aussi des notation plus raccourcies par exemple pour symboliser les nombres décimaux

```
const charDecimal = /\d/;
```

ps : on peut utiliser les regex dans d'autres langages (côté serveur par exemple) mais aussi directement côté HTML dans un attribut pattern (cela rajoute un couche de sécurité).

### 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 l'exercice suivant nous allons mettre en place une page web qui comporte un formulaire (1 input pour le mail et un pour le mot de passe, et une div dans laquelle on affichera des messages d'erreur)

Sur l'input du mail on va surveiller le clavier et utiliser la regex du mail (expliquée précédemment) ainsi que la fonction .test()

Si le mail a un format valide Alors on affiche l'input en Vert sinon le mail n'est pas valide et on affiche l'input en rouge

Sur l'input du mot de passe nous allons contrôler la longueur du mot de passe (il doit être compris entre 6 et 8 caractères) auquel cas cela affiche des messages d'erreurs correspondantes. Ensuite sur cet input nous allons utiliser les regex charDecimal et charSpecial ainsi que la fonction match(), si le mot de passe ne match pas la regex charDecimal on affiche un message à l'utilisateur, on va faire pareil pour la regex charSpecial.

Si l'utilisateur à un mot de passe valide on affiche un message de confirmation

Login  Password

Le password est :

- trop court
- doit contenir un chiffre
- doit contenir un caractère spécial \$,&,@ ou !

Login  Password

Le password est valide