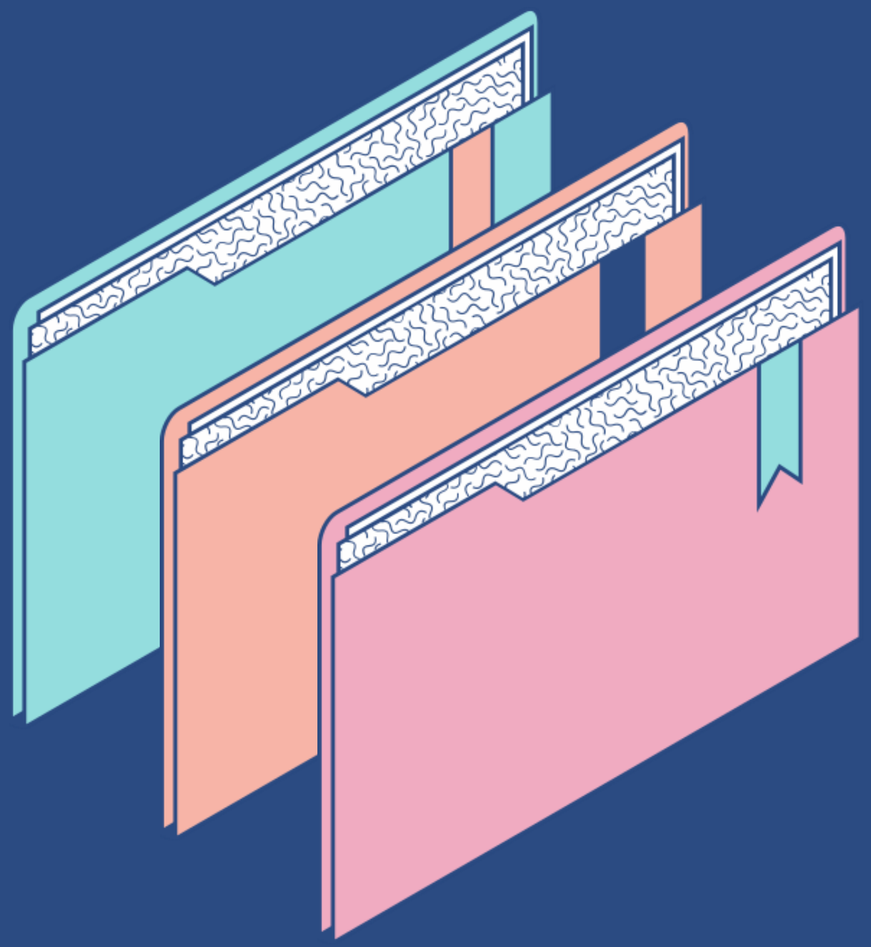
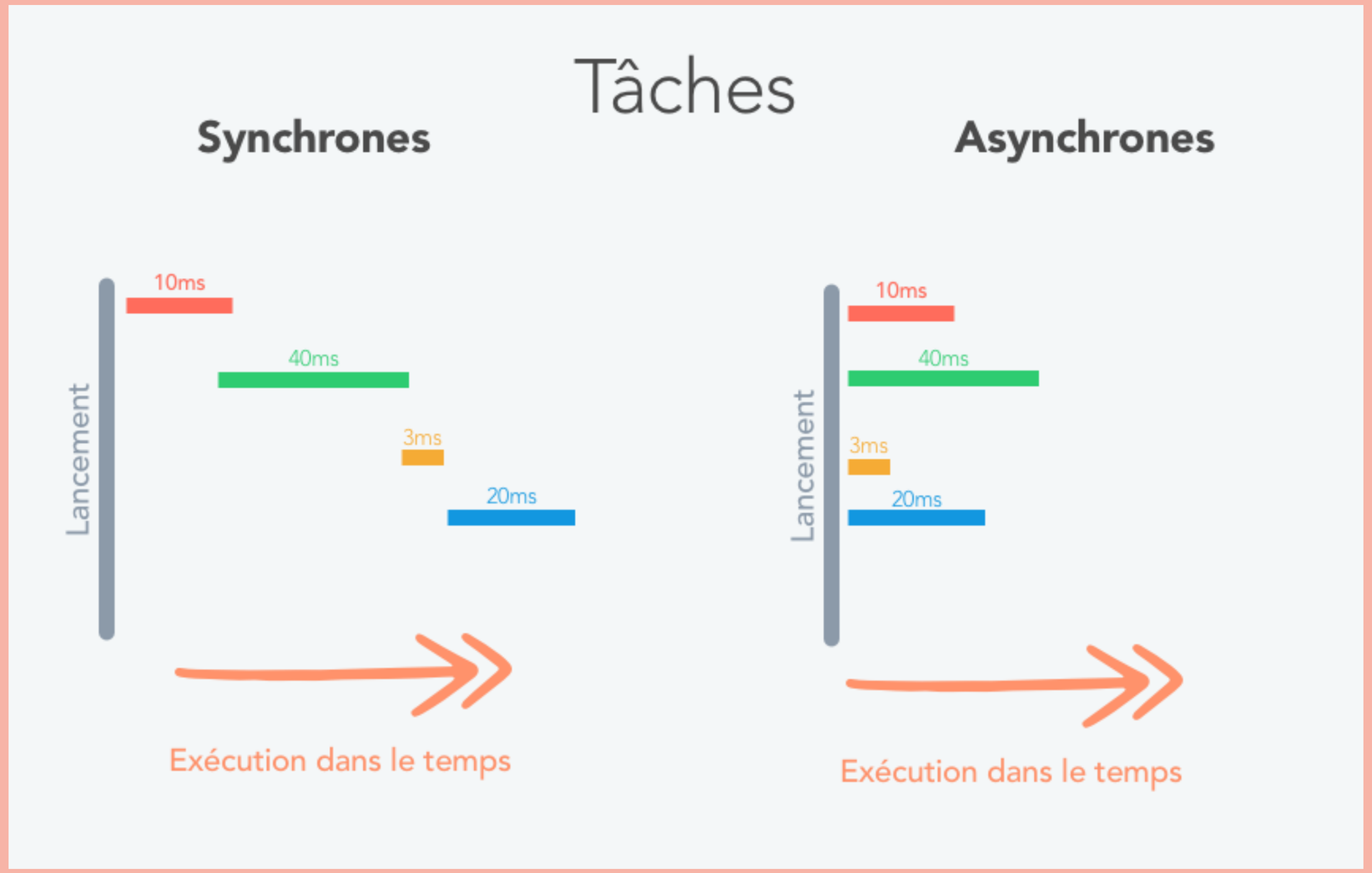
Javascript


JavaScript
<script>
if(age > 19){
alert("Adult");
} else{
alert("Teenager");
}
</script>

SOMMAIRE



Asynchrone


Tâches
Synchrones
Asynchrones
Lancement
10ms
40ms
3ms
20ms
Exécution dans le temps
Lancement
10ms
40ms
3ms
20ms
Exécution dans le temps

Introduction asynchrone

Des opérations sont définies comme asynchrones si elles s'effectuent en parallèle.
à contrario, des opérations sont synchrones si elles s'exécutent l'une après l'autre.

Par défaut, le JS est un langage synchrone, qui ne s'exécute que sur 1 thread.
Chaque opération attendra donc la fin de la précédente pour s'exécuter

Promesses JS

Une promesse JS est un objet représentant l'état d'une opération asyncrhone
(en cours; terminée avec succès; stoppée par échec)

```
const promesse = new Promise((resolve, reject) => {
    //Tâche asynchrone à réaliser
    /*Appel de resolve() si la promesse est résolue (tenue)
     *ou
     *Appel de reject() si elle est rejetée (rompue)*/
});
```

Promesses JS

```
function loadScript(src){
    return new Promise((resolve, reject) => {
        let script = document.createElement('script');
        script.src = src;
        document.head.append(script);
        script.onload = () => resolve('Fichier ' + src + ' bien chargé');
        script.onerror = () => reject(new Error('Echec de chargement de ' + src));
    });
}

const promesse1 = loadScript('boucle.js');
```

Gestion succès / erreurs

Promesses JS

Le résultat d'une promesse se récupère à l'aide d'un bloc : then / catch.
Si la promesse est tenue, le code contenu dans le bloc "then" s'exécute.
Si elle n'est pas tenue, le bloc "catch" s'exécute.

```
await answer().then(response => {
    console.log("succès :) "+response);
}).catch(e => {
    console.log("Erreur :'( " +e);
});
```

async JS

le mot clé "async" devant une déclaration de fonction permet de la transformer en asynchrone en créant un objet Promise.

Le mot clé "await" est uniquement valable au sein des fonctions async.
Permet d'attendre que la promesse soit résolue ou rejetée avant d'exécuter la suite.

```
async function test(){
    const promise = new Promise((resolve, reject) => {
        setTimeout(() => resolve('Ok !'), 2000)
    });

    let result = await promise; //Attend que la promesse soit résolue ou rejetée
    alert(result);
}
```

Exercice d'application : Async functions

Créez une première fonction nommée fileUpload(fileSize);
Cette fonction simulera l'upload d'un fichier.
Au bout de 3 secondes, la fonction retournera :

- resolve si fileSize est inférieur à 10
- reject si fileSize est supérieur à 10

Créez une seconde fonction result(), qui affiche :

- "Fichier chargé avec succès !" si la promesse est résolue
- "Erreur de chargement du fichier !" si la promesse n'est pas résolue

Présentation du JSON

Le JSON est un format d'échange de données léger et performant très utilisé.

Il est construit par rapport à 2 structures :

- Une collection de paires nom/valeur
- Une liste ordonnée de valeurs

```json
{
    "prenom": "Pierre",
    "nom": "Giraud",
    "adresse": {
        "rue": "30 Impasse des Lilas",
        "ville": "Toulon",
        "cp": 83000,
        "pays": "France"
    },
    "mails": [
        "pierre.giraud@edhec.com",
        "pierre@pierre-giraud.com"
    ]
}
```

Utilisation JSON en JS

Le JS possède 2 méthodes particulièrement intéressantes pour manipuler les fichiers JSON :

- parse() qui analyse une chaine JSON et en construit une valeur JS
- stringify() qui convertit une valeur JS en chaine JSON.

```
//Objet JavaScript
let utilisateur = {
    "prenom": "Pierre",
    "nom": "Giraud",
    "adresse": {
        "rue": "30 Impasse des Lilas",
        "ville": "Toulon",
        "cp": 83000,
        "pays": "France"
    },
    "mails": [
        "pierre.giraud@edhec.com",
        "pierre@pierre-giraud.com"
    ]
  };

//Conversion en chaine JSON
let json = JSON.stringify(utilisateur);

document.getElementById("resultat").innerHTML
=
    "Type de la variable : " + typeof(json)
+ "<br>Contenu de la variable : " + json;
```

Présentation AJAX

L'AJAX est un ensemble de techniques permettant d'envoyer et récupérer des données vers et depuis un serveur de façon asynchrone.

L'AJAX se base sur l'objet XMLHttpsRequest, disponible sur tous les navigateurs, et permettant de faire des requêtes HTTP en JS.

Créer requêtes AJAX

Pour effectuer une requêtes AJAX, nous utiliserons toujours 4 étapes :

- Création d'un objet XMLHttpRequest
- Initialisation de la requête
- Envoi de la requête
- Gestion des événements pour prise en charge réponse serveur

Créer requêtes AJAX

L'initialisation se fait grace à open(a,b,c,d,e) sur l'objet XMLHttpRequest où :

a correspond au type de requête : GET / POST

b représente l'URL de destination de la requête

c (facultatif) définit le caractère asynchrone (true par défaut)

d / e (facultatif) définit nom d'utilisateur et mdp si authentification.

Créer requêtes AJAX

Une fois la requête initialisée par open(), on spécifie le format de la réponse souhaitée au sein de la propriété responseType :

- "" OU "text": réponse en chaine de caractère (par défaut)
- "arraybuffer" : renvoi un objet ArrayBuffer
- "blob" : renvoi un objet blob
- "document" : renvoi un document XML
- "json" : renvoi un fichier JSON.

Dans la majorité des cas, le type de retour attendu est le json

Créer requêtes AJAX

cours.html | JS cours.js

```
1   //On crée un objet XMLHttpRequest
2   let xhr = new XMLHttpRequest();
3
4   //On initialise notre requête avec open()
5   xhr.open("GET", '/renseigner/une/url');
6
7   //On veut une réponse au format JSON
8   xhr.responseType = "json";
9
10  //On envoie la requête
11  xhr.send();
```

Créer requêtes AJAX

Une fois la requête envoyée au serveur, nous devons réceptionner sa réponse.

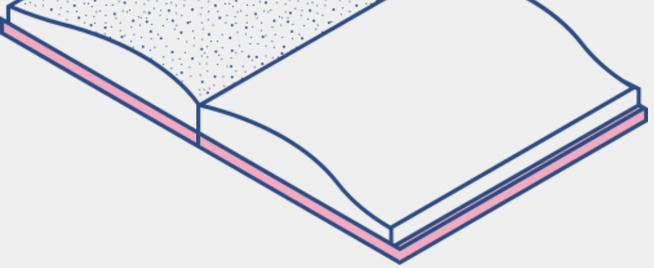
Pour cela, nous utiliserons 3 évènements spécifiques :

- load : se déclenche lorsque la requête a bien été effectuée
- error : se déclenche lorsque la requête n'a pas aboutit
- progress : se déclenche à intervalles réguliers et définit où en est la requête.

Créer requêtes AJAX

Le statut de l'évènement load nous permettra de connaître de quelle façon s'est terminée la requête.

Dans la pratique, nous testerons si le code de retour est bien égal à 200, signifiant que la requête s'est bien terminée sur un succès.



Créer requêtes AJAX

```
//On crée un objet XMLHttpRequest
let xhr = new XMLHttpRequest();

//On initialise notre requête avec open()
xhr.open("GET", "une/url");

//On veut une réponse au format JSON
xhr.responseType = "json";

//On envoie la requête
xhr.send();

//Dès que la réponse est reçue...
xhr.onload = function(){
    //Si le statut HTTP n'est pas 200...
    if (xhr.status != 200){
        //...On affiche le statut et le message correspondant
        alert("Erreur " + xhr.status + " : " + xhr.statusText);
    //Si le statut HTTP est 200, on affiche le nombre d'octets téléchargés et la réponse
    }else{
        alert(xhr.response.length + " octets  téléchargés\n" + JSON.stringify(xhr.response));
    }
};

//Si la requête n'a pas pu aboutir...
xhr.onerror = function(){
    alert("La requête a échoué");
};

//Pendant le téléchargement...
xhr.onprogress = function(event){
    //lengthComputable = booléen; true si la requête a une length calculable
    if (event.lengthComputable){
        //loaded = contient le nombre d'octets téléchargés
        //total = contient le nombre total d'octets à télécharger
        alert(event.loaded + " octets reçus sur un total de " + event.total);
    }
};
```