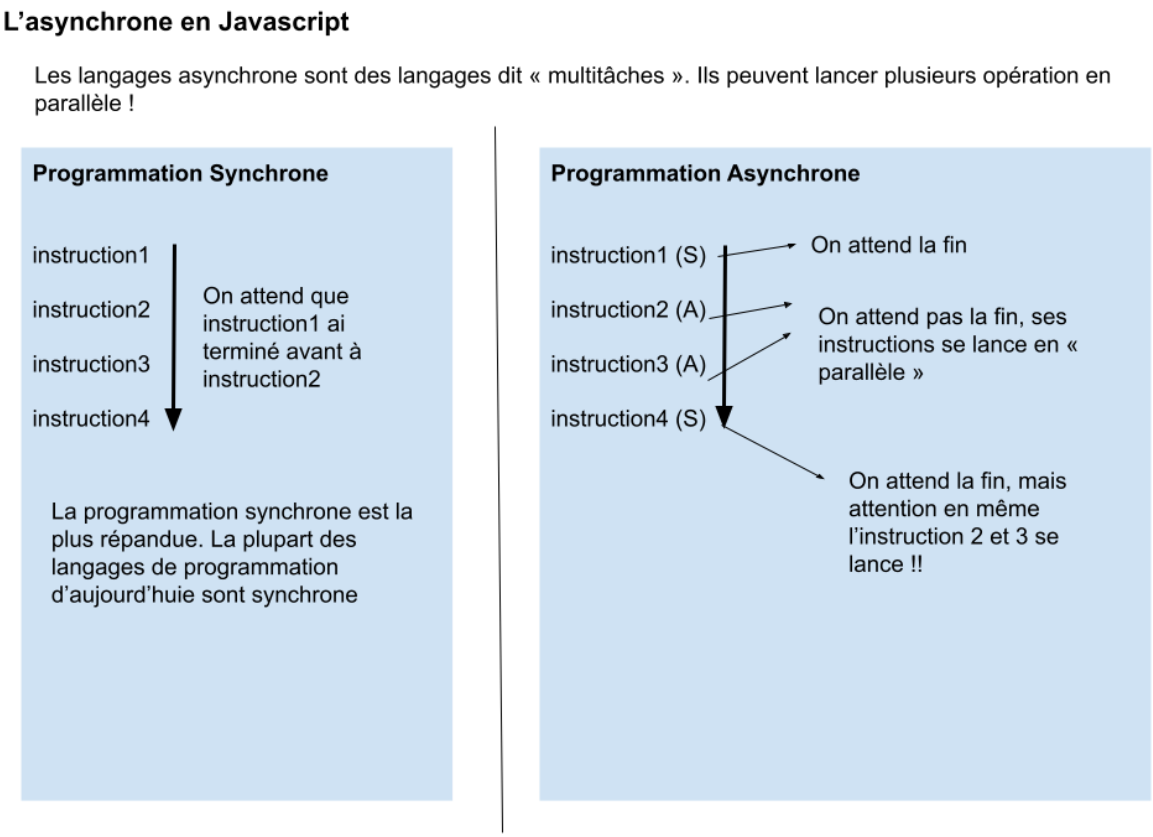
L'asynchrone

L'asynchrone est la possibilité de pouvoir lancer plusieurs ligne de codes en « même temps » : On parle de « parralélisme »



La `Promise`

En javascript il éxiste un objet qui permet de contrôller l'asynchrone et aussi de la représenter programmaticalement. C'est la `Promise` !

Tout d'abord il faut comprendre qu'il éxiste des données qui sont synchrone, qui se manipule diréctement dans notre programme. C'est la cas de la plupart des données.

Cependant d'autres données ne sont pas encore résolue. Prenons un exemple, je souhaiterais lancer en même temps

- Récupérer le titre de la page
- Additionner 145 + 89

Ici, nous allons manipuler des promesses (promise).

```
// Exemple de code « synchrone »
const titre = recupereTitrePage()
const resultat = addtionner(145, 89)

console.log(titre) // Type : string !
console.log(resultat) // Type : number !

// Exemple de code « asynchrone »
const titre = recupereTitrePageAsync()
const resultat = additionnerAsync(145, 89)

console.log(titre) // type: Promise
console.log(resultat) // type: Promise
```

Manipuler les `Promise`

Nous savons que un code asynchrone retourne toujours une `Promise` ! C'est grâce à cet objet, que nous allons pouvoir manipuler « le temps » :

```
// Ici on récupérer le titre de la page de manière
// asynchrone. C'est à dir que le reste du code
// s'éxécutera en même temps que cette fonction
const promise = recupereTitreAsync()

console.log('Autre instruction ...')
console.log('Autre instruction 2 ...')

// Pour récupérer et attendre le résultat d'une promesse
// on utilise la onction `.then`
promise.then(titre => {
  // C'est seulement dans le `.then` que nous pouvons récupérer
  // la valeur asynchrone !
  console.log(titre)
})

// Il est possible de que notre promesse échoue, qu'une erreur se
// produise lors de la récupération du titre. De base, les erreurs
// contenue dans les promesse n'impact aucunement le reste du code,
// on dit que la promesse est « isolé »
// Par contre, la fonction `.then` ne sera jamais appelé, cependant
// nous pouvons récupérer la possible erreur en utilisant `.catch`
promise.catch(error => {
  console.log('Oups une erreur est survenue lors de la récup. du titre')
  console.error(error)
})

// Il est possible d'enchainer les deux :
promise
  .then(titre => {
    console.log(titre)
  })
  .catch(error => {
    console.log('Oups ...')
    console.error(error)
```

```
})
```

Créer une `Promise`

Il est possible de créer nos propres promesse (des valeurs prennant un certains temps avant de se résoudre !)

Pour cela, nous allons utiliser notre première `class`. Une `class` est un objet javascript un peut particulier. Il se **construit** avec le mot clef **new** :

De base, une promesse ne se **résoud jamais** :/ ...

```
const myPromise = new Promise()

myPromise.then(() => {
  // Ici le console log ne s'éxécutera jamais ...
  // de base une promesse ne se termine jamais :/
  console.log('terminé ...')
})
```

Pour qu'un promesse puissent de résoudre nous devons lui envoyer une fonction d'initialisation.

Cette fonctions accépte 2 paramètres :

- `resolve` : Fonction à appeler lorsque la promesse est terminé
- `error` : Fonction à appeler si quelque chose tourne au drames

```
// Je créer une promesse avec une fonction d'initialisation
const myPromise = new Promise((resolve, error) => {
  console.log('Ma promesse se lance')

  // J'utilise setTimeout  afin d'attendre 3 secondes
  window.setTimeout(() => {
    // Fonciton se lancant après 3 seconds
    // On termine notre promesse en appelant :
    // Soit resolve (tout c'est bien passé)
    // Soit error (Il y a eu une erreur)
    resolve()
  }, 3000)
})

console.log("En même temps que l'initialisation ...")

// On attend la fin de la promesse :
myPromise.then(() => {
  // Cette fonction est lancé après 3 secondes !
  console.log('Coucou')
})
```

Attendre plusieurs `Promise`

Il est possible d'utiliser la fonction `Promise.all` afin d'attendre plusieurs promesse en une fois !

Exemple, je souhaiterais lancer les opérations asynchrones suivante :

- Attendre 10s et afficher coucou
- Attendre 8s et afficher salut
- Attendre 15s et afficher ça vas ?

```
// On attend 10s et on affiche coucou (asynchrone)
const p1 = waitAndDisplay(10, 'coucou') // Promise
const p2 = waitAndDisplay(8, 'salut') // Promise
const p3 = waitAndDisplay(15, 'ça va ?') // Promise

// On attend plusieurs promesse à la fois :
Promise.all([p1, p2, p3]).then(() => {
  console.log('Se lance lorsque toutes les promesse sont terminé !!!!')
})
```

Les promesses et les valeurs de retours

La plupart des promesses retourne une valeur. Ici, nous avons vu comment créer des promesse, attendre mais pas comment retourner des valeurs.

Pour cela, on utilise la fonction `resolve` vu plus haut on spécifié la valeur à rretourner :

```
// Je créer une promesse avec une fonction d'initialisation
const myPromise = new Promise((resolve, error) => {
  console.log('Ma promesse se lance')

  // J'utilise setTimeout  afin d'attendre 3 secondes
  window.setTimeout(() => {
    // Fonciton se lancant après 3 seconds
    // On termine notre promesse en appelant :
    // Soit resolve (tout c'est bien passé)
    // Soit error (Il y a eu une erreur)
    // Ici, on spécifie une valeur à notre promesse "terminé"
    resolve('terminé')
  }, 3000)
})

console.log("En même temps que l'initialisation ...")

// On attend la fin de la promesse :
myPromise.then(valeur => {
  // Ici la paramètre « valeur » contient « "Terminé" »
  console.log(valeur) // "Terminé"
  // Cette fonction est lancé après 3 secondes !
  console.log('Coucou')
})
```

Utiliser `async` et `await`

Depuis `2020`, javascript à beaucoup évolué et c'est énormément popularisé et simplifier. Nous avons la possibilité de simplifier tout le fonctionnement vu plus haut à l'aide de 2 mots clefs : `async` et `await` :

Exemple Synchrone

```
function additionner(x, y) {
  return x + y
}

const resultat = additionner(10, 5)
```

Exemple Asynchrone

```
// On ajoute async devant le fonction
async function additionner(x, y) {
  return x + y
}

console.log(additionner(x, y)) // Promise !
```

N'importe quelle fonction peut devenir asynchrone en utilisant le mot clef `async` juste devant `function`. Dès lors la fonction retournera une `Promise`

Il est aussi possible, dans une fonction asynchrone d'utiliser `await` :

```
async function additionner(x, y) {
  return x + y
}

async function soustraire(x, y) {
  return x - y
}

async function multiplier(x, y) {
  return x * y
}

async function diviser(x, y) {
  return x / y
}

async function calculer() {
  // Ici dans une fonciton asynchrone
  // nous pouvons attendre une promesse (
  // la rendre asynchrone) en utilisant `await`
  const resultat = await additionner(10, 56)

  return resultat
}
```