ECMAScript 6/7

ES6 / ES2015

ECMAScript 5 est officielle depuis 2009

Il s'agit de la norme décrivant la syntaxe JavaScript

- ECMAScript 6 est officielle depuis 2015. Cette version est complètement intégrée dans les navigateurs récents. On peut la considérer comme le standard
- ECMAScript 7 ou 2016 est sortie en 2016
- ECMAScript 8 ou 2017 est sortie en 2017
- ECMAScript 9 ou 2018 est sortie en 2018
- ECMASCRIPT 10 ou 2019 est sortie en 2019
- Liens
  - Compatibilités Javascripts
  - Mozilla Developer

Les nouveautés de l'ES6

Mots-clefs de variables

let et var

```
// `var` limite la portée à la  fonction, peu importe les blocs
for ( var i=0; i<5; i++) {
// ...
}
console.dir(i); // 5
// `let` limite la portée au bloc
for ( let i=0; i<5; i++) {
// ...
}
console.dir(i); // undefined
```

Le mot clé const

```
const foo = 'bar';
foo = 'test'; // TypeError: Assignment to constant variable
```

Pour des raisons de simplicité et de bonnes pratiques, nous utiliserons principalement `let` pour définir nos variables et `const` pour les constantes.

Exercice de compréhension

Durant un laps de temps relativement court. Environ 10 minutes, tester les propriétés de ces mots clés.

- Tester les différents cas d'utilisation des mots `let`, `const` et `var` pour observer leurs différences.
- Affecter une valeur dans une boucle
- Tester la valeur en cours de boucle, après la boucle...
- Réaffecter une valeur à `const`
- Remarquez les messages d'erreurs constatés et appropriez vous l'utilisation de la console.

Les templates litéraux

- Source

```
let name = 'Toto';
const template = `<h1>Bonjour ${name}</h1>
  <p>Vous avez ${18 + 20} ans</p>`;
console.log(template);
// '<h1>Bonjour Toto</h1>\n\t<p>Vous avez 38 ans</p>
```

Cette écriture possède différents avantages :

- On peut facilement concaténer et interpoler
- Les retours à la ligne ne posent plus de problèmes

Exercice de compréhension

- Tester d'utiliser un template litéral.
- Concatener du texte avec une variable
- Effectuer une opération
- Appeler une fonction

Les promesses

Les Promises est une nouvelle syntaxe permettant de simplifier l'utilisation de nombreux callbacks

L'objet Promise (pour « promesse ») est utilisé pour réaliser des traitements de façon asynchrone. Une promesse représente une valeur qui peut être disponible maintenant, dans le futur voire jamais.

Vous avez déjà du cotoyer ce fonctionnement lorsque vous avez abordé l'ajax

```
const maPremierePromesse = new Promise((resolve, reject) => {
  // réaliser une tâche asynchrone et appeler :
```

```
  // resolve(uneValeur); // si la promesse est tenue
  // ou
  // reject("raison d'echec"); // si elle est rompue
});

const myPromise = new Promise(function(resolve, reject) {
  setTimeout(function() {
    resolve('foo');
  }, 300);
});
myPromise.then(function(value) {
  console.log(value);
}).catch(function(error) {
  // Appelé lorsque "reject" est appelé
  console.log(error);
})
```

Les fonctions flechées

Les fontions fléchées ou arrow function permettent de définir une fonction de façon raccourcie. Cela permet aussi d'éviter de lier cetains mots-clefs comme this, super ou des arguments.

```
let fn = (name, age = 40) => {
  return `<h1>Bonjour ${name}</h1><p>Vous avez ${age} ans</p>`;
}
fn('Toto', 35);
let fn2 = name => `Bonjour ${name}`;
fn2('Toto');
```

Cette syntaxe raccourcie permet également de définir des fonctions anonymes.

```
fetch('./data.json')
  .then(res => res.json())
  .then(console.table);
fetch('./data.json')
  .then(res =res.json())
  .then((data) => {
    for ( let row of data) {
      console.dir(row);
    }
  });
```

Exercice de compréhension

Exercice de 10 minutes.

- Tester d'utiliser une fonction fléchée
- Concever un event addEventListner
- Manipuler setTimeout

Solution > exercices/ecmascript/arrowFunctions.html

Les modules

ES6 permet de fonctionner avec une approche modulaire. Cela permet par exemple de cloisonner chaque classe et valeur, sans définir de variables globales.

```
export const name = 'Toto'
export var age = 35;
export default class Person {} // Export par défault
```

On utilise le mot `export` afin de rendre disponible un élément L'export par défaut permet plus tard d'importer une valeur sans connaitre son nom

```
import { name, age as ageDuCapitaine } from './test.module.js';
// Importer des valeurs membres d'un module en utilisant un alias
import Test from './test.module.js';
// Importer la valeur par défaut d'un module
import Test, { name } from './test.module.js';
// Importer la valeur par défaut et une valeur nommée
```

On utilise le mot `import` afin de récupérer un élément qui a été exporté

Exercice de compréhension

Durée : 15 minutes

- Concevez un fichier Javascript exportant un module
- Importez le module dans un autre fichier
- Testez les différentes écritures

Exemple > exercices/ecmascript/index.html
à ouvrir avec un php -S localhost:3030 par exemple.

Spread et Rest opérateurs

Le paramètre `...` permet d'utiliser un tableau comme argument d'un fonction, simplifiant l'utilisation d'un nombre indéfini d'arguments.

```
function maFonction(premier,...leReste) {
  console.log("premierArgument", premier);
  console.log("leReste", leReste);
}
maFonction("un", "deux", "trois");
```

La syntaxe du `Spread operator` permet de passer rapidement toutes les propriétés d'un élément itérable (comme un tableau ou un objet)

```
const array = [1,2,3];
console.log(...array); // Output 1,2,3
```

Exemple > exercices/ecmascript/arrowFunctions.html

Async & Await

Une fonction `async` est une fonction qui retourne une Promise. Pour attendre le retour de la Promise il faut utiliser le mot-clé `await`.

```
function resolveAfter2Seconds() {
  return new Promise(resolve ={
    setTimeout(() ={
      resolve('resolved');
    }, 2000);
  });
}
async function asyncCall() {
  var result = await resolveAfter2Seconds();
  console.log(result); // expected output: 'resolved'
}
asyncCall();
```

Solution > exercices/ecmascript/arrowFunctions.html

Les tableaux

Les tableaux sont une des structures les plus utiles et les plus populaires dans l'eco-systeme JavaScript.

Mais en plus d'être capable de gérer une multitude de cas d'utilisation, ils sont surtout très puissant via l'ensemble des fonctionnalites de l'API `array`.

Toutes les méthodes suivantes sont fonctionnelles nativement sur les navigateurs. On peut les regrouper en plusieurs catégories :

- Transformation

    - filter
    - map
    - flatMap
- Interaction
    - reverse
    - reduce[Right]
    - find[Index]
    - sort
- Iteration
    - forEach
- Autre

filter

Comme son nom l'indique, cette méthode permet de filtrer un tableau pour ne conserver que les éléments validant la condition du filtre

```
const arr = [1,2,3,4];
const evenArr = arr.filter(num => num % 2 === 0); // [2,4]
```

map

Possiblement une des fonctions les plus utilisées actuellement. Elle permet d'itérer sur un tableau et d'appliquer une transformation sur chaque élément, suivant la règle passée en paramètre

```
const arr = [1,2,3,4];
const oneMoreArr = arr.map(num => num + 1); // [2,3,4,5]
```

flat

Cette fonction permet d'applatir un tableau ayant des sous tableaux. La fonction peut prendre un paramètre, ce dernier etant le niveau que l'on veut atteindre

```
const arr = [1,[2,[3,[4]]]];
const flatten = arr.flat(2); //[1,2,3,[4]]
```

reverse

Cette fonction renverse l'ordre des éléments dans le tableau

```
const arr = [1,2,3,4];
arr.reverse(); // [4,3,2,1]
```

reducer

Cette fonction va effectuer une opération sur chaque valeur puis accumuler le resultat dans une variable

```
const arr = [1,2,3,4];
const sum = arr.reduce((acc, num) => acc + num); // 10
```

find et findIndex

Cette méthode prend en paramètre une fonction de verification, et retourne le premier élément qui valide la condition (ou undefined si aucun).

```
const arr = [5, 12, 8, 130, 44];
const matched = arr.find(num => num% 2 === 0); // 12
const matchingIndex = arr.findIndex( (element) => element > 13 ); // 3
```

sort

Comme son nom l'indique, cette méthode permet de trier un tableau. Par defaut, le tri se fait sur une string en utf-16, dans les autres cas il faut fournir une fonction permettant le tri.

```
const arr = [5, 12, 8, 130, 44];
arr.sort((num1, num2) => {
  return num1-num2;
});
console.log(arr); //  [ 5, 8, 12, 44, 130 ]
// Attention . Array.sort() modifie le tableau source
```

forEach

Cette fonction itère sur chaque élément du tableau, puis éxécute la fonction passée en argument pour chaque élément.

```
const arr = [1,2,3,4];
arr.forEach((num) => {
  console.log(num); // 1/2/3/4
})
```