JS

1. Rappels
2. Environnement de debug
3. ES6+
4. **POO**
5. jQuery
6. Expressions régulières
7. Échange de données
8. APIs HTML5

LA VIELLE SYNTAXE ES5


```
function Character( firstName, lastName ) {
    this.firstName = firstName; // propriétés
    this.lastName = lastName;
}
Character.prototype.fullName = function() { // méthode
    return this.firstName + ' ' + this.lastName;
}

// héritage
function Salamanca( firstName ){
    Character.call(this, firstName, 'Salamanca' ); // super
}
Salamanca.prototype = new Character(); // héritage
var uncle = new Salamanca( 'Hector' );
console.log( uncle.fullName() );
```

Notes :

Exemple : https://codepen.io/uidlt/pen/gOrJVRx?editors=0011

POO : CLASS

```
class Character {
    firstName = 'unknown'; // propriété publique "firstName" (ES2021 ?)
    lastName; // undefined

    constructor(firstName, lastName) { // constructeur de la classe
        this.firstName = firstName; // on écrase la valeur initiale
        this.lastName = lastName;
    }
    fullName(){ // déclaration de méthode
        return `${this.firstName} ${this.lastName}`;
    }
}
const heisenberg = new Character('Walter', 'White');
console.log( heisenberg.firstName ); // accès à la propriété
console.log( heisenberg.fullName() ); // appel de la méthode
```

Notes :

https://codepen.io/uidlt/pen/mdPZdbg?editors=0011

Avec la nouvelle syntaxe ES6, il est désormais possible de déclarer une classe à l'aide du mot clé class. Tout ce qui est contenu dans une classe est exécuté en mode strict.

La méthode constructor permet d'initialiser de nouvelles instances. Il ne peut y avoir qu'un seul constructeur par classe et il est facultatif.

La création de propriétés d'instance se fait habituellement dans le constructeur à l'aide du mot clé `this`.

On peut aussi déclarer des propriétés en dehors du constructeur comme montré dans l'exemple ci-dessus mais il s'agit d'une syntaxe qui n'est pas encore dans la spec officielle : cette syntaxe est en stage 3 de spécification c'est à dire l'avant dernier niveau avant l'intégration dans la spec officielle.

On ne sait pas encore si cette syntaxe a des chances d'être intégrée ou pas dans la version ES12/ES2021 de la spec

cf. https://github.com/tc39/proposal-class-fields et https://tc39.github.io/proposal-class-fields/

POO : HÉRITAGE


```
class Salamanca extends Character {
    isDead = false; // propriété supplémentaire
    constructor( firstName ){
        super( firstName, 'Salamanca' ); // constructeur parent
        if ( this.firstName === 'Tuco' ) {
            this.isDead = true;
        }
    }
    fullName() { // override de la méthode fullName()
        const result = super.fullName(); // appel de la méthode parente
        return result.toUpperCase();
    }
}
const uncle = new Salamanca('Hector');
console.log( uncle.fullName() ); // hérite des méthodes de Character
```

Notes :

Grâce au mot clé extends, l'héritage se construit de manière plus lisible qu'avec les prototypes.

Dans le constructeur de la classe fille, la méthode super() permet d'appeler le constructeur de la classe parente. super() doit obligatoirement être appelée avant d'utiliser le mot clé "this".

super peut également servir à appeler une méthode parente (`super.fullName()` dans notre exemple)

POO : STATIC

```
class Counter {
    static counter = 0; // propriété statique (ES2021 ?)
    static getCounter() { // méthode statique (ES2021 ?)
        return this.counter++;
    }
}

console.log(
    Counter.getCounter(),    // 0
    Counter.counter,         // 1
    Counter.getCounter(),    // 1
    Counter.counter,         // 2
);

const c = new Counter();
console.log(c.getCounter()); // Error: c.getCounter is not a function
```

Notes :

https://codepen.io/uidlt/pen/MWyMWyo?editors=0011

On peut déclarer une propriété statique ou une méthode statique avec le mot clé `static`. Notez qu'en déclarant une variable avec this dans une méthode statique, celle ci sera disponible en tant qu'attribut statique.

Les méthodes statiques ne sont pas accessibles via les instances de classes. Elles sont généralement utilisées pour créer des fonctions utilitaires.

Cette notation n'est pas encore dans la spec EcmaScript officielle, mais est en cours de spécification dans une feature actuellement en stage 3 (intégration possible dans ES12/ES2021 ?) : https://github.com/tc39/proposal-static-class-features/ et https://tc39.github.io/proposal-static-class-features/