

# DÉVELOPPEMENT JAVASCRIPT

*jour n°2*

# JS

# JS

- 3. La POO en JS
- 4. Les modules
- 5. Le typage

# JS

1. Rappels

2. Premiers pas en JS

## **3. La POO en JS**

4. Les modules

5. Le typage

6. API DOM

7. Ajax

8. jQuery

“ EN JAVASCRIPT,  
ON PEUT PAS  
FAIRE  
DE VRAIE  
***P00*** ”

# 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 Targaryen( firstname ){
    Character.call(this, firstname, 'Targaryen' ); // super
}
Targaryen.prototype = new Character(); // héritage
var jonsnow = new Targaryen( 'Aegon' );
console.log( jonsnow.fullname() );
```

---

Notes :

<https://codepen.io/kumquats/pen/MEzJzp?editors=0011> (<https://codepen.io/kumquats/pen/MEzJzp?editors=0011>)

# POO AVEC ES6

1. déclaration de class
2. propriétés
3. méthodes
4. getter/setter
5. private
6. static

# POO AVEC ES6 : CLASS

```
class Animal {  
  constructor(name) { // Constructeur de la classe  
    console.log('Nouvelle instance', name);  
  }  
}  
  
const threeEyedRaven = new Animal('Bran');
```

---

Notes :

<https://codepen.io/kumquats/pen/XgpLqj?editors=0011> (<https://codepen.io/kumquats/pen/XgpLqj?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.

# POO AVEC ES6 : PROPRIÉTÉS <sup>2.8</sup>

```
class Animal {  
  name = 'unknown'; // propriété publique "name" (ES2019 ?)  
  
  constructor(name) {  
    this.name = name; // modification de la propriété "name"  
  }  
}  
  
const threeEyedRaven = new Animal('Bran');  
console.log( threeEyedRaven.name ); // accès à la propriété "name"
```

---

Notes :

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

La possibilité de déclarer des propriétés en dehors du constructeur comme montré dans l'exemple ci-dessus 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 ES10/ES2019 de la spec

cf. <https://github.com/tc39/proposal-class-fields> (<https://github.com/tc39/proposal-class-fields>) et <https://tc39.github.io/proposal-class-fields/> (<https://tc39.github.io/proposal-class-fields/>)



# POO AVEC ES6 : MÉTHODES

```
class Animal {  
  name = 'unknown';  
  
  constructor(name) {  
    this.name = name;  
  }  
  fly() { // déclaration de méthode  
    console.log(`${this.name} is flying !`);  
  }  
}  
  
const threeEyedRaven = new Animal('Bran');  
threeEyedRaven.fly(); // appel de méthode
```

# POO AVEC ES6 : PRIVATE

```
class Animal {
  name = 'unknown';
  #canFly = false; // propriété privée (#)
  constructor(name) {
    this.name = name;
    this.#canFly = (name == 'Bran');
  }
  fly() {
    console.log(`${this.name} is flying !`);
  }
}

const threeEyedRaven = new Animal('Bran');
threeEyedRaven.fly();
```

---

## Notes :

Le support des propriétés et méthodes privées est en stage 3 de spécification. Ce n'est donc pas encore dans la spec EcmaScript officielle. Néanmoins il est possible de les utiliser grâce à au plugin Babel : [@babel/plugin-proposal-class-properties](https://babeljs.io/docs/en/babel-plugin-proposal-class-properties) (<https://babeljs.io/docs/en/babel-plugin-proposal-class-properties>) .

Si vous vous demandez pourquoi on écrit `#propriete` et pas `private propriete` comme dans d'autres langages, la réponse se trouve ici : [https://github.com/tc39/proposal-class-fields/blob/master/PRIVATE\\_SYNTAX\\_FAQ.md#why-arent-declarations-private-x](https://github.com/tc39/proposal-class-fields/blob/master/PRIVATE_SYNTAX_FAQ.md#why-arent-declarations-private-x) ([https://github.com/tc39/proposal-class-fields/blob/master/PRIVATE\\_SYNTAX\\_FAQ.md#why-arent-declarations-private-x](https://github.com/tc39/proposal-class-fields/blob/master/PRIVATE_SYNTAX_FAQ.md#why-arent-declarations-private-x))

# POO AVEC ES6 : GET/SET

```
class Animal {
  name = 'unknown';
  #canFly = false;
  constructor(name) {
    this.name = name;
    this.#canFly = (name == 'Bran');
  }
  fly() {
    console.log(`${this.name} is flying !`);
  }
  get canFly() {
    return this.#canFly;
  }
}
const threeEyedRaven = new Animal('Bran');
console.log( threeEyedRaven.canFly );
```

# POO AVEC ES6 : HÉRITAGE<sup>2.12</sup>

```
class Dragon extends Animal {
  isDead = false;
  constructor( name ){
    super( name ); // constructeur parent
    if ( this.name == 'Viserion' ) {
      this.isDead = true;
    }
  }
  fly() { // override de la méthode fly()
    super.fly(); // appel de la méthode parente fly
    console.log(`${this.name} burns everything !`);
  }
}
const dragon = new Dragon('Rhaegal');
dragon.fly(); // dragon hérite des méthodes de Animal
```

---

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.fly() dans notre exemple)

# POO AVEC ES6 : STATIC

```
class Counter {
  static counter = 0; // propriété statique (ES10 ?)
  static getCounter() { // méthode statique (ES10 ?)
    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/kumquats/pen/bRgzqY?editors=0011> (<https://codepen.io/kumquats/pen/bRgzqY?editors=0011>)

On peut déclarer une propriété statique ou une méthode statique avec le mot clé `static`. A noter 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 ES10/ES2019 ?) : <https://github.com/tc39/proposal-static-class-features/> (<https://github.com/tc39/proposal-static-class-features/>) et <https://tc39.github.io/proposal-static-class-features/> (<https://tc39.github.io/proposal-static-class-features/>)

# JS

1. Rappels
2. Premiers pas en JS
3. La POO en JS

## **4. Les modules**

5. Le typage
6. API DOM
7. Ajax
8. jQuery

“ EN JAVASCRIPT,  
ON PEUT PAS  
ORGANISER  
SON CODE  
***PROPREMENT*** ”

# MODULES : PROBLÉMATIQUE <sup>3.3</sup>

```
<!-- Sans utilisation des modules -->  
<script src="header.js"></script>  
<script src="menu.js"></script>  
<script src="breadcrumb.js"></script>  
<script src="footer.js"></script>  
<script src="main.js"></script>
```



```
<!-- Avec utilisation des modules -->  
<script type="module" src="main.js"></script>
```

---

## Notes :

Le système de modules permet de gérer plus facilement les dépendances entre les différents fichiers de notre application. Au lieu de lister à plat l'ensemble des fichiers requis par notre application (dans l'ordre adéquat !) c'est le fichier JS principal qui indique les fichiers dont il dépend, qui eux même indiquent les fichiers dont ils dépendent et ainsi de suite. Ce mécanisme est particulièrement utile lorsque l'on développe des applications utilisant de nombreux fichiers et notamment des librairies tierces.



# MODULES : PRINCIPE

## *hodor.js*

```
const character = 'Hodor';  
export default character;
```

## *main.js*

```
import character from "./hodor.js";  
console.log( character ); // 'Hodor'
```

---

### Notes :

Le principe des modules est de diviser le code JavaScript en plusieurs fichiers appelés "modules". Toutes les variables contenues dans un modules ne sont disponibles qu'au sein de celui-ci. Il est en revanche possible d'exporter certaines variables afin de les exposer au reste de l'application. Un module peut faire appel à d'autres module grace à l'instruction "import". Il pourra ainsi avoir accès à toutes les variables exportées par le module importé.

# MODULES : EXPORTS MULTIPLES

## *hodor.js*

```
const character = 'Hodor';
const what = 'Door';
export default character;
export const message = 'Hold the ' + what;
```

## *main.js*

```
import character from './hodor.js';
import { message, what } from './hodor.js';
console.log( message, what ); // 'Hold the door', undefined
```

---

Notes :

Un module peut exporter plusieurs valeurs, mais une seule peut être l'export par défaut.

Pour exporter des valeurs supplémentaires, il suffit d'utiliser l'instruction `export` suivi de la valeur, sans le mot clé `default`.

Au niveau des imports, il y a là aussi une différence : il faut entourer le nom de la (ou des) valeur qu'on veut importer avec des accolades :

```
import { maValeur } from './monmodule.js';
```

On peut tout à fait importer plusieurs valeurs avec une seule instruction `import`, il faut alors séparer les valeurs par des virgules :

```
import { maValeur1, maValeur2 } from './monmodule.js';
```

Si l'on veut récupérer à la fois l'export par défaut et d'autres exports avec une seule instruction alors on peut écrire :

```
import maValeurParDefaut, { maValeur1, maValeur2 } from
'./monmodule.js';
```

# MODULES + CLASSES

 */animals/Animal.js*

```
export default class Animal { ... }
```



 */animals/Dragon.js*

```
import Animal from './Animal.js';  
export default class Dragon extends Animal { ... }
```



 */main.js*

```
import Dragon from './animals/Dragon.js';  
const d = new Dragon('Rhaegal');
```

# JavaScript modules via script tag - LS

Usage

% of all users

Global

79.9% + 0.49% = 80.38%

3.7

Loading JavaScript module scripts using `<script type="module">`  
Includes support for the `nomodule` attribute.

Current aligned

Usage relative

Date relative

Apply filters

Show all

?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Blackberry Browser	Opera Mobile *	Chrome for Android	Firefox for Android	IE Mobile	U f
	12-14	2-53	4-59	3.1-10	10-46	3.2-10.2								
	15	54-59	60	10.1	47	10.3								
6-10	16-17	60-64	61-71	11-11.1	48-56	11-11.4		2.1-4.4.4	7	12-12.1			10	
11	18	65	72	12	57	12.1	all	67	10	46	71	64	11	
		66-67	73-75	TP										

<

>

Notes

Sub-features (1)

Known issues (2)

Resources (13)

Feedback

1

 Support can be enabled via `about:flags`

2

 Support can be enabled via `about:config`

4

 Does not support the `nomodule` attribute

5

 A **polyfill** is available for Safari 10.1/iOS Safari 10.3

6

`nomodule` scripts are not executed, but they're still fetched

Can I use...

Browser support tables for modern web technologies

Created & maintained by [@Fyrd](#), design by [@Lencs](#).

Support data contributions by the [GitHub community](#)

Support via Patreon

Become a caniuse Patron to support the site and disable ads for only \$1/month!

Site links

[Home](#)

[Feature index](#)

[Browser usage table](#)

Legend

■ = Supported

■ = Not supported

■ = Partial support

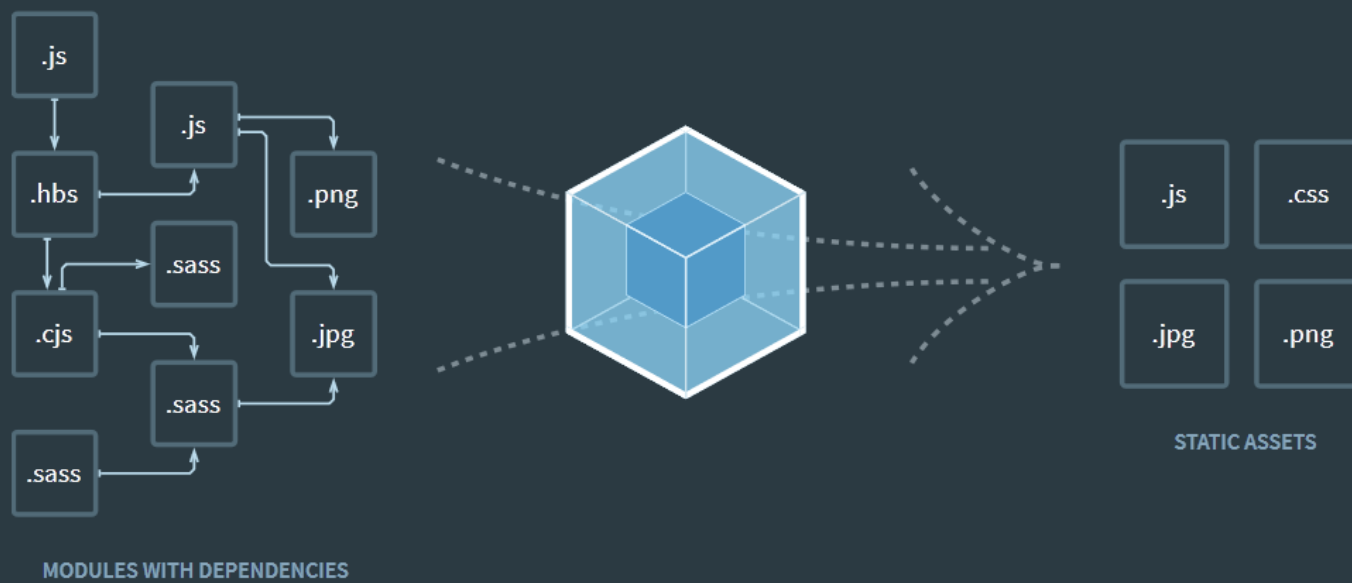
Notes :

Support navigateur des modules ES6 : <https://caniuse.com/#feat=es6-module> (<https://caniuse.com/#feat=es6-module>)

Le support est pour le moment très partiel. Difficile d'entrevoir une utilisation des modules pour une application grand public.

mais...

## bundle your scripts



## Write your code

src/index.js

```
import bar from './bar';  
  
bar();
```

src/bar.js

```
export default function bar() {  
  //  
}
```

Notes :

<https://webpack.js.org/> (<https://webpack.js.org/>) webpack permet de regrouper dans un seul fichier (le "bundle") l'ensemble des fichiers nécessaires au fonctionnement de l'application et donc d'étendre le support navigateur au delà des navigateurs qui supportent le système de modules.

Webpack permet d'intégrer différents type de "loaders".

Les loaders sont des filtres qui sont appliqués aux fichiers avant qu'ils ne soient ajoutés dans le bundle.

Le loader le plus fréquemment utilisé est `babel-loader` qui permet de s'assurer que les modules contenant de l'ES6 seront bien compilés en ES5.

# JS

1. Rappels
2. Premiers pas en JS
3. La POO en JS
4. Les modules

## **5. Le typage**

6. API DOM
7. Ajax
8. jQuery

“ EN JAVASCRIPT,  
ON PEUT PAS  
FAIRE  
DE  
***TYPAGE STATIQUE*** ”

# TYPAGE FAIBLE & DYNAMIQUE

- faiblement typé : conversion de type implicite
- dynamique : une variable peut changer de type

```
let maVariable = 'chaîne';  
maVariable = 42;  
maVariable = function(){ return 1 };  
  
console.log( maVariable + 1337 ); // "function(){return 1}1337"
```





flow

[Getting Started](#) [Docs](#) [Try](#) [Blog](#)



4.4

# FLOW IS A STATIC TYPE CHECKER FOR JAVASCRIPT.

**GET STARTED**

**INSTALL FLOW**

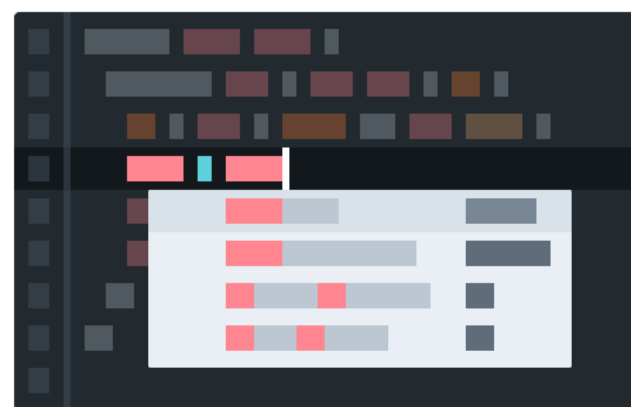
Star

18,861

Current Version: **v0.92.1**

## CODE FASTER.

Tired of having to run your code to find bugs? Flow identifies problems as you code. Stop wasting your time guessing and checking.



Notes :

<https://flow.org/> (<https://flow.org/>)



# flow

- Permet le typage statique en JavaScript
- CLI qui affiche les erreurs de typage
- maintenu par Facebook

Notes :

Flow est un programme capable de vérifier le typage (pas de compiler) et peut s'intégrer avec Babel.

```
npm install --save-dev @babel/preset-flow flow-bin  
./node_modules/.bin/flow init
```

Fichier *.babelrc* :

```
{  
  "presets": [ "@babel/env", "@babel/preset-flow" ]  
}
```

# FLOW : EXEMPLE

```
// @flow
// la ligne ci-dessus est nécessaire pour que Flow teste le fichier

function stringify( n: number ): ?string {
  if( n > 0 ) {
    return n.toString();
  }
}
```


---

## Notes :

Dans l'exemple ci-dessus on indique que la fonction "stringify" retourne un string. Le point d'interrogation indique que la fonction peut également retourner un résultat vide.

*Une fois le typage ajouté aux fichiers JS, on peut lancer la vérification à l'aide de la commande :*

```
./node_modules/.bin/flow
```


flow
4.7

```

const a:number = 12;
const stringify = (value/*:number*/):string =>
return value.toString();

console.log( stringify( a ) );
console.log( stringify( false ) );

```

Errors
JSON
AST

6: console.log( stringify( false ) );
^ Cannot call `stringify` on a non-callable value

References:
6: console.log( stringify( false ) );
^ [1]
2: const stringify = (value/\*:number\*/):string =>
^ [2]

Notes :

Testeur en ligne de la syntaxe flow :

<https://flow.org/try/#0MYewdgzgLGBAhgLjAVwLYCMCmAnGBeGARgCYBuAKFEImmWeswBzOgMwE98YAKANzgBtkmA>  
<https://flow.org/try/#0MYewdgzgLGBAhgLjAVwLYCMCmAnGBeGARgCYBuAKFEImmWeswBzOgMwE98YAKANzgBtkmA>

# FLOW : LES TYPES

## TYPES PRIMITIFS

```
// @flow

const text: string = "Text"           // Chaîne de caractère
const count: number = 42              // Nombre
const isBoolean: boolean = true       // Booléen
const empty: null = null              // Null
const notDefined: void = undefined    // Undefined
```

## TYPES LITTÉRAUX

```
// @flow

// "value" n'accepte que les valeurs
// "up", "down", "left" et "right"
function setDirection(value: "up"|"down"|"left"|"right") {
    // ...
}
```

## TYPES MULTIPLES

```
// @flow

// "phoneNumber" accepte soit une chaîne de caractère
// soit un nombre
function setPhoneNumber(phoneNumber: string|number ) {
    // ...
}
```

## TYPES DE TABLEAUX

```
// @flow

let numberArray: Array<number> = [];

numberArray.push(42); // 👍
numberArray.push("42"); // ☹ Erreur !
```

## TYPES DE CLASSES

```
// @flow

class Animal { /* */ }
class Dragon extends Animal { /* */ }
```

```
class Dragon extends Animal { /* */ }
```

```
class IceDragon extends Dragon { /* */ }
```

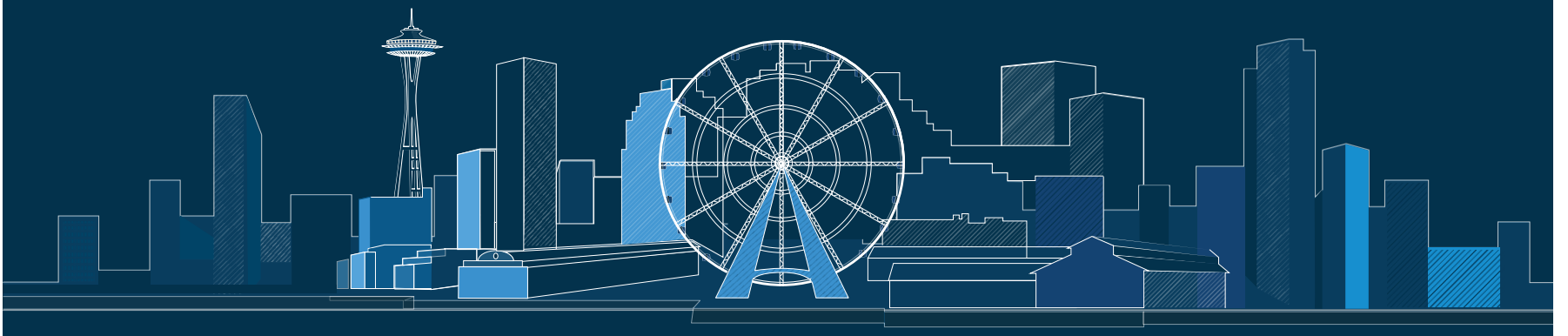
```
let viserion:Dragon = new Dragon(); // 👍
```

```
viserion = new IceDragon(); // 👍
```

```
viserion = new Animal(); // ❌ Erreur !
```

# TYPESCRIPT

- superset de JavaScript
- type, interfaces, abstract, ...
- compilateur TS -> ES3
- maintenu par Microsoft



# TYPESCRIPT : EXAMPLE

```

class Animal {
  readonly name:string = 'unknown';
  private _canFly:boolean = false;
  constructor(name:string) {
    this.name = name;
    this._canFly = (name == 'Bran');
  }
  fly():void {
    console.log(`${this.name} is flying !`);
  }
  get canFly():boolean{
    return this._canFly;
  }
}

class Dragon extends Animal {
  public isDead:boolean = false;
  constructor( name:string ){
    super( name );
    if ( this.name == 'Viserion' ) {
      this.isDead = true;
    }
  }
  fly():void {
    super.fly();
    console.log(`${this.name} burns everything !`);
  }
}

const threeEyedRaven:Animal = new Animal('Bran');
console.log( threeEyedRaven.canFly );

const dragon:Dragon = new Dragon('Rhaegal');
dragon.fly();

```

Notes :

## Version Flow

(<https://flow.org/try/#0FAYwNghgzIAECCA7AlgWwmWBvYBIREqApgFxQAuATsogOawC8sA5AK6IDWiA9gO6LMA3HgDEgxMzABCIBLMjnoAvngaspaOJABu3MgAJi64+oZQykTeYNy0lgAGACRYHl6+MemwXpqyDrAAhI1peJm4tETksOJSOXmK1N15BsjDd8piZO0EcNRqLMkcFdMPGsiABRGREYoAJQgesQJFe6EwTEQRF4CBQnssiWSAlGCuqyoaROtdodztd3j2shxwFA>)

## Version TypeScript

([https://www.typescriptlang.org/play/index.html#src=class%20Animal%20%7B%0D%0A%09readonly%20name%3Astring%3D%20%27Bran%27%0D%0A%09private%20\\_canFly%3Aboolean%3D%20false%0D%0A%09constructor\(name%3Astring\)%20%7B%0D%0A%09this.name%20%3D%20name%3B%0D%0A%09this.\\_canFly%20%3D%20\(name%20%3D%20%27Bran%27\)%0D%0A%09}%0D%0A%09fly\(\)%3Avoid%20%7B%0D%0A%09console.log\(`\\${this.name}%20is%20flying%20!`\)%0D%0A%09}%0D%0A%09get%20canFly\(\)%3Aboolean%20%7B%0D%0A%09return%20this.\\_canFly%3B%0D%0A%09}%0D%0A%09}%0D%0A%09const%20threeEyedRaven%3AAnimal%20%3D%20new%20Animal\(%27Bran%27\)%3B%0D%0A%09console.log\(threeEyedRaven.canFly\)%3B%0D%0A%09const%20dragon%3ADragon%20%3D%20new%20Dragon\(%27Rhaegal%27\)%3B%0D%0A%09dragon.fly\(\)%3B%0D%0A%09%2F%3E](https://www.typescriptlang.org/play/index.html#src=class%20Animal%20%7B%0D%0A%09readonly%20name%3Astring%3D%20%27Bran%27%0D%0A%09private%20_canFly%3Aboolean%3D%20false%0D%0A%09constructor(name%3Astring)%20%7B%0D%0A%09this.name%20%3D%20name%3B%0D%0A%09this._canFly%20%3D%20(name%20%3D%20%27Bran%27)%0D%0A%09}%0D%0A%09fly()%3Avoid%20%7B%0D%0A%09console.log(`${this.name}%20is%20flying%20!`)%0D%0A%09}%0D%0A%09get%20canFly()%3Aboolean%20%7B%0D%0A%09return%20this._canFly%3B%0D%0A%09}%0D%0A%09}%0D%0A%09const%20threeEyedRaven%3AAnimal%20%3D%20new%20Animal(%27Bran%27)%3B%0D%0A%09console.log(threeEyedRaven.canFly)%3B%0D%0A%09const%20dragon%3ADragon%20%3D%20new%20Dragon(%27Rhaegal%27)%3B%0D%0A%09dragon.fly()%3B%0D%0A%09%2F%3E))



TypeScript 3.3 is now available. [Download](#) our latest version today!

Select...

TypeScript

Share

Options

Run

JavaScript

```
1 class Animal {
2     static count:number = 0;
3
4     readonly name:string;
5     public isDead:boolean = false;
6     private canFly:boolean = false;
7
8     constructor( name:string, canFly:boolean = false) {
9         this.name = name;
10        this.canFly = canFly;
11        Animal.count++;
12    }
13
14    getName():string{
15        return this.name;
16    }
17
18    fly() {
19        if (!this.canFly) {
20            this.isDead = true;
```

```
1 var __extends = (this && this.__extends) || (function () {
2     var extendStatics = function (d, b) {
3         extendStatics = Object.setPrototypeOf ||
4             ({ __proto__: [] } instanceof Array && function (d, b) { for (var p in b) if (b[p] !== undefined) __extends(d, b); });
5     };
6     return function (d, b) {
7         extendStatics(d, b);
8         function __() { this.constructor = d; }
9         d.prototype = b === null ? Object.create(b) : __.prototype;
10    };
11 })();
12
13 var Animal = /** @class */ (function () {
14     function Animal(name, canFly) {
15         if (canFly === void 0) { canFly = false; }
16         this.isDead = false;
17         this.canFly = false;
18         this.name = name;
19         this.canFly = canFly;
20     }
```

Notes :

TypeScript dispose d'un testeur en ligne, sur <https://www.typescriptlang.org/play>  
(<https://www.typescriptlang.org/play>)

MERCI !