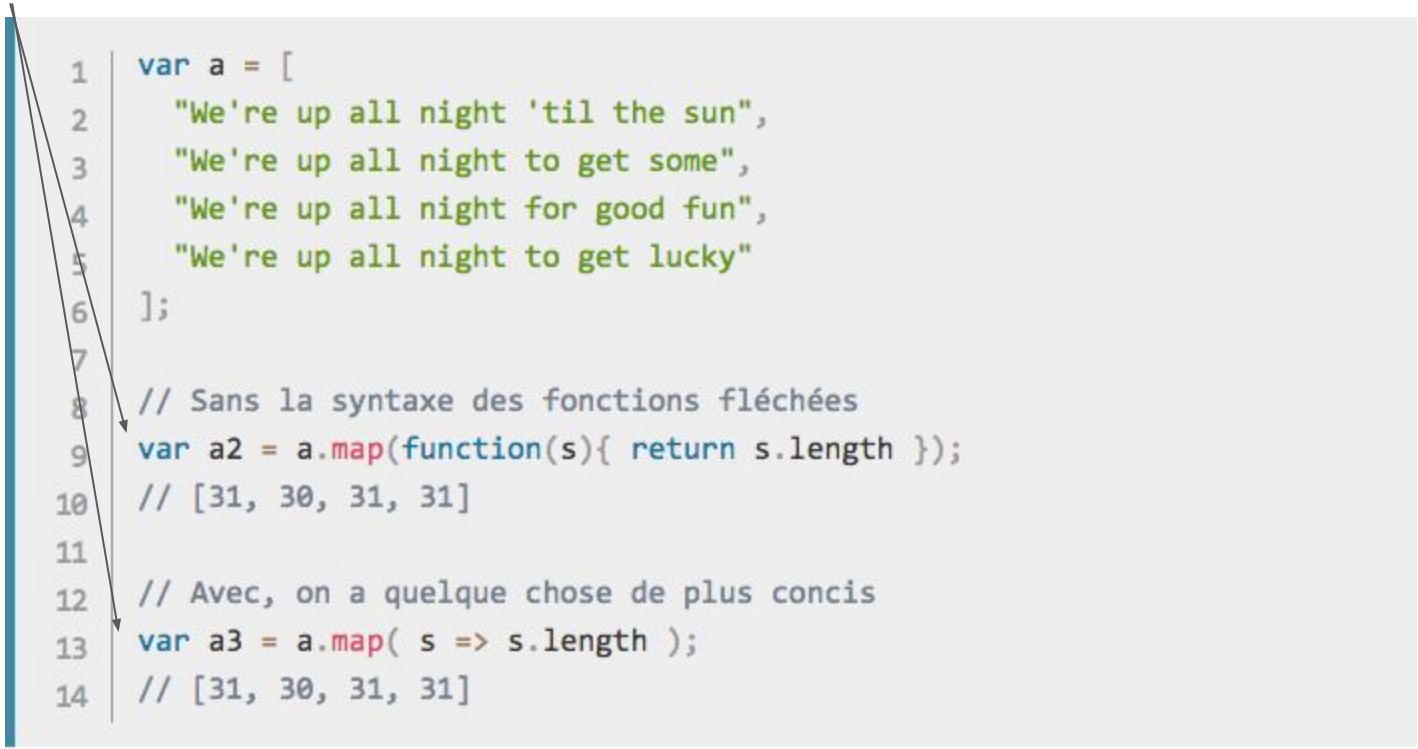
Les fonctions fléché capte et fixe le this du scope parent, là ou elle a été déclarer.

Les fonctions fléchées

Syntaxe plus courte:

Pour des aspects fonctionnels, la légèreté de la syntaxe est bienvenue .
par exemple :



```
1  var a = [
2    "We're up all night 'til the sun",
3    "We're up all night to get some",
4    "We're up all night for good fun",
5    "We're up all night to get lucky"
6  ];
7
8  // Sans la syntaxe des fonctions fléchées
9  var a2 = a.map(function(s){ return s.length });
10 // [31, 30, 31, 31]
11
12 // Avec, on a quelque chose de plus concis
13 var a3 = a.map( s => s.length );
14 // [31, 30, 31, 31]
```

```
// FONCTION CLASSIC ----------------------------------
function myClassicFunction(){
    return "Hello my friend";
};

console.log(myClassicFunction());
```

Fonction classic

```
// FONCTION FLECHE SANS ARGUMENT ---------------------
const myArrowFunction = () => {
    return "Hello my friend"
};

console.log(myArrowFunction());
```

Fonction flèche sans argument

```
// FONCTION FLECHE AVEC 1 SEUL ARGUMENT----------------
const myArrowFunction1Argument = prenom => {
    return "Je m'appel "+ prenom + "!";
}

console.log(myArrowFunction1Argument("Matt"));
```

Fonction flèche avec 1 seul argument

```
// FONCTION FLECHE AVEC PLUSIEURS ARGUMENTS -----------
const myArrowFunctionArguments = (number1, number2) =>{
    return number1 - number2;
}

console.log(myArrowFunctionArguments(10,5));
```

Fonction avec plusieurs argument

```
//FONCTION FLECHE EN RETOUR D'UNE AUTRE FONCTION

const me = {
  name: "Matt",
  present1: function (b){
    var a = () => this.name
    return a(b);
  }
};

console.log(me.present1());
```

Fonction fléché dans une méthode d’objet:

Le fait que la méthode fléché soit dans une méthode classique le this de la fonction fléchée ne créer pas de nouveau contexte, son this prend la valeur du contexte englobant la fonction fléché, c’est à dire la valeur de la méthode de l’objet en question qui elle même prendra la valeur de l’objet quand elle sera invoqué par celui ci.

Problème

```
1  function Personne() {
2    // Le constructeur Personne() définit `this` comme lui-même.
3    this.age = 0;
4
5    setInterval(function grandir() {
6      // En mode non strict, la fonction grandir() définit `this`
7      // comme l'objet global et pas comme le `this` defini
8      // par le constructeur Personne().
9      this.age++;
10   }, 1000);
11 }
12
13 var p = new Personne();
```

Avec ECMAScript ⅗, ce problème a pu être résolu en affectant la valeur de ‘this’ à une autre variable.

```
1  function Personne() {
2    var that = this;
3    that.age = 0;
4
5    setInterval(function grandir() {
6      // La fonction callback se réfère à la variable `that`
7      // qui est le contexte souhaité
8      that.age++;
9    }, 1000);
10 }
```

Pas de ‘this’ lié à la fonction:

Jusqu’a l’apparition des fonctions fléchées, chaque nouvelle fonction définissait son propre ‘this’(un nouvelle objet dans le cas d’un constructeur, ‘undefined’ dans les appels de fonctions stricts, le contexte de l’objet si la fonction est appelée comme une méthode, etc..) Cela a pu entraîner des confusions lorsque’on utilisait un style de programmation orientée objet.

Les fonctions fléchées ne créent pas de nouveau contexte, elles utilisent la valeur ‘this’ de leur contexte, c’est a dire le this de la valeur du parent englobant la fonction fléché.
Alors que les fonction classique crée un nouveau contexte a chaque fois.

Le code qui suit fonctionne de la façon attendue:

```
1  function Personne(){
2    this.age = 0;
3
4    setInterval(() => {
5      this.age++;
6      // |this| fait bien référence
7      // à l'objet personne
8    }, 1000);
9  }
10
11 var p = new Personne();
```

Dans une fonction fléché étant donné que ‘this’ provient du contexte englobant, si on invoque une fonction via la méthode ‘call’ ou ‘apply’, cela ne passera que des arguments mais n’aura aucun effet sur ‘this’.

Dans le cas si dessous ‘this’ se réfère uniquement au contexte englobant c’est a dire l’objet “ajouter”.

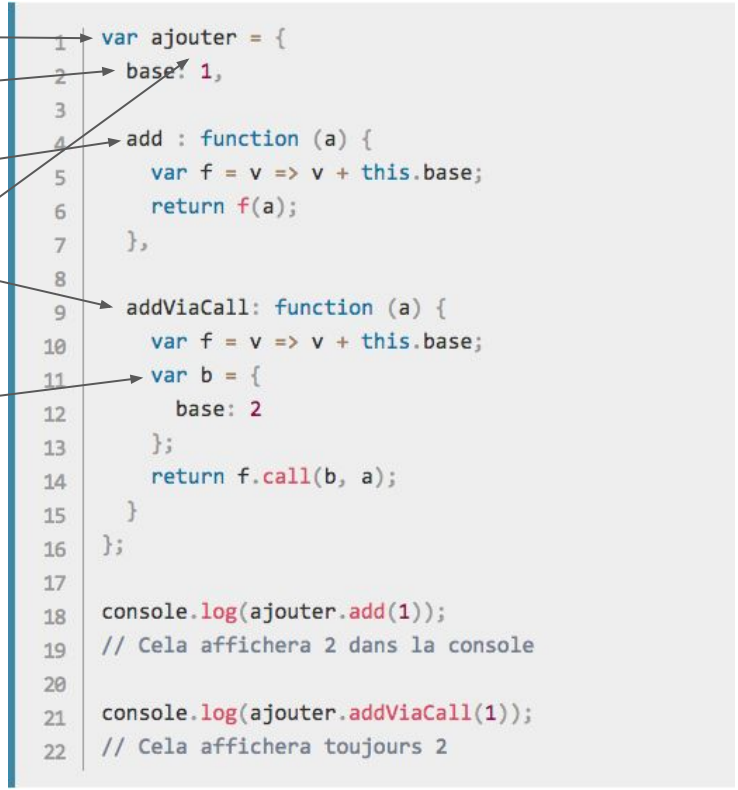
Objet.

Propriété.

Fonction.

this.base est insensible au contexte de cet fonction.

Car il se base sur le contexte qui l’englobe.



```
var ajouter = {
  base: 1,

  add : function (a) {
    var f = v => v + this.base;
    return f(a);
  },

  addViaCall: function (a) {
    var f = v => v + this.base;
    var b = {
      base: 2
    };
    return f.call(b, a);
  }
};

console.log(ajouter.add(1));
// Cela affichera 2 dans la console

console.log(ajouter.addViaCall(1));
// Cela affichera toujours 2
```

Les fonctions fléchées n'exposent pas d'objet arguments : arguments.length, arguments[0], arguments[1], et autres ne font donc pas référence aux arguments passés à la fonction fléchés. Dans ce cas arguments est simplement une référence à la variable de même nom si elle est présente dans la portée englobante :

```
var arguments = 42;
var arr = () => arguments;

arr(); // 42

function toto() {
  var f = (i) => arguments[0] + i;
  // lien implicite avec arguments de toto
  return f(2);
}

toto(1); // 3
```

Les fonctions fléchées n'ont donc pas leur propres arguments, mais la plupart des cas, les paramètres du reste représentent une bonne alternative :


```
function toto() {
  var f = (...args) => args[0];
  return f(2);
}

toto(1); // 2
```

Les fonctions fléchées comme méthodes:

Comme indiqué précédemment, les fonctions fléchées sont mieux indiquées pour les fonctions qui ne sont pas des méthodes. Prenons un exemple pour illustrer ce point :

```
'use strict';
var obj = {
  i: 10,
  b: () => console.log(this.i, this),
  c: function() {
    console.log(this.i, this);
  }
}

obj.b();
// affiche undefined, Window (ou l'objet global de l'environnement)

obj.c();
// affiche 10, Object {...}
```

Les fonctions fléchées ne possèdent pas de prototype :

```
1 | var Toto = () => {};
2 | console.log(Toto.prototype);
```

Utiliser le mot-clé ‘yield’ : Le mot-clé yield ne peut pas être utilisé dans le corps d'une fonction fléchée (sauf si cela intervient dans une autre fonction, imbriquée dans la fonction fléchée). De fait, les fonctions fléchées ne peuvent donc pas être utilisées comme générateurs.

Utiliser le mot-clé new : Les fonctions fléchées ne peuvent pas être utilisées comme constructeurs et lèveront une exception si elles sont utilisées avec le mot-clé new.