ARROW FUNCTIONS

Les fonctions fléchées, introduites dans ECMAScript 6 (ES6), sont un moyen concis d'écrire des fonctions en JavaScript. Elles ont une syntaxe plus courte par rapport aux expressions de fonction traditionnelles et ne lient pas leur propre **`this`**, **`arguments`**, **`super`** ou **`new.target`**.

SYNTAXE

```
(param1, param2, ..., paramN) => { instructions }
```

Si le corps de la fonction contient seulement une seule expression, vous pouvez omettre les accolades et le mot-clé **`return`** :

```
(param1, param2, ..., paramN) => expression
```

EXEMPLES

1. Fonction fléchée de base :

```
const add = (a, b) => {
    return a + b;
};
```

2. Sans paramètres :
Si une fonction fléchée n'a pas de paramètres, utilisez des parenthèses vides :

```
const greet = () => 'Hello, world!';
```

3. Un seul paramètre :
Si une fonction fléchée a un seul paramètre, vous pouvez omettre les parenthèses :

```
const square = x => x * x;
```

DIFFÉRENCES PRINCIPALES

SYNTAXE

Fonctions classiques :
utilisent le mot-clé `**function**`

```
function add(a, b) {
    return a + b;
}
```

Fonctions fléchées :
ont une syntaxe plus courte

```
const add = (a, b) => a + b;
```

CONSTRUCTEUR

Fonctions classiques :
peuvent être utilisées comme constructeurs.

```
function Person() {}
const p = new Person(); // Fonctionne
```

Fonctions fléchées :
ne peuvent pas être utilisées comme constructeurs, et un appel avec `**new**` provoque une erreur.

```
const Person = () => {};
const p = new Person(); // Erreur :
          Person n'est pas un constructeur
```

CONTEXTE `THIS`

Fonctions classiques :
ont leur propre **`this`**, qui dépend de la façon dont la fonction est appelée.

```
    function Person() {
        this.age = 0;
        setInterval(function() {
            this.age++; // `this` se réfère à
l'objet global ou est indéfini en mode strict
        }, 1000);
    }
```

Fonctions fléchées :
n'ont pas leur propre **`this`**. Elles héritent de **`this`** du contexte lexical environnant. Cela les rend particulièrement utiles dans les situations où vous devez stocker valeur **`this`** à partir d'une fonction

```
    function Person() {
        this.age = 0;
        setInterval(() => {
            this.age++; // `this` se réfère
                à l'instance de Person
        }, 1000);
    }
```

OBJET `ARGUMENTS`

Fonctions classiques :
ont leur propre objet **`arguments`**, qui contient tous les arguments passés à la fonction.

```
    const showArgs() {
    console.log(arguments); // `arguments`
                            est défini
    };
    showArgs(1, 2, 3); // [1, 2, 3]
```

Fonctions fléchées :
n'ont pas leur propre objet **`arguments`.**

```
    const showArgs = () => {
    console.log(arguments); // `arguments`
                                n'est pas défini
    };
    showArgs(1, 2, 3); // Erreur
```

HOISTING (ÉLÉVATION)

Fonctions classiques :
sont élevées.

```
console.log(add(2, 3)); // 5
funstion add(a, b){
    return a + b;
}
```

Fonctions fléchées :
ne sont pas élevées.

```
console.log(add(2, 3)); // Erreur : add
                        n'est pas définie
const add = (a, b) => a + b;
```

CONCLUSION

Les fonctions fléchées fournissent un code plus compact et plus lisible, en particulier lors de l'utilisation de fonctions imbriquées et de callbacks. Cependant, il est important de se rappeler leurs caractéristiques, telles que propre **`this`** et des **`arguments`.**