02 - Fonctions utilisateurs, contextes d'exécution et scopes

Définir une fonction

Il existe 4 syntaxes pour écrire une fonction en Javascript :

1. Déclaration de fonction
2. Expression de fonction
3. Fonction fléchée
4. Syntaxe objet avec `new Function()`

1. Déclaration de fonction

La syntaxe traditionnelle est la **déclaration de fonction** avec l'instruction `function` :

Exemple

```
function calculAire(iLargeur, iLongueur) {
    return iLargeur * iLongueur;
}

const iAire = calculAire(5, 6);

console.log("Aire : "+iAire); // Affiche 30
```

2. Expression de fonction

Le mot-clé `function` permet de définir une fonction à l'intérieur d'une expression (c'est-à-dire marquée de l'opérateur égal) : on parle alors d'expression de fonction.

Exemple 1 : expression de fonction sans passage d'argument

```
const hello = function() {
    console.log("Hello world!");
};

// Appel de la fonction
hello();
```

Exemple 2 : expression de fonction avec passages d'arguments

```
const calculAire = function(iLargeur, iLongueur) {
    return iLargeur * iLongueur;
};

let iAire = calculAire(5, 6);

console.log("Aire : "+iAire); // Affiche 30
```

3. Fonctions fléchées (= arrow functions)

Une (expression de) fonction fléchée - *arrow function* en anglais - permet en premier lieu d'avoir une syntaxe plus courte que les expressions de fonctions abordées au point 2.

Il existe plusieurs syntaxes. Celle de base est la suivante :

```
const nom_de_la_fonction = (arguments) => ( corps de la fonction; );
```

La partie "corps de la fonction" peut aussi bien être encadrée par des accolades au lieu des parenthèses :

```
const nom_de_la_fonction = (arguments) => { corps de la fonction; };
```

ou même, si le corps de la fonction de comporte qu'une seule (ligne d') instruction(s), ne pas être encadrée du tout :

```
const nom_de_la_fonction = (argument 1, argument 2...) => corps de la fonction;
```

Exemple

```
const calculAire = (iLargeur, iLongueur) => { return iLargeur * iLongueur; };
```

Quand il n'y a qu'un seul argument (après le signe égal), il n'est pas nécessaire de l'encadrer avec des parenthèses :

```
const add = x => return x + 1;
```

Enfin, si la fonction ne reçoit aucun argument, il convient alors de l'indiquer avec des parenthèses vides :

```
const bonjour = () => return "bonjour !";
```

4. Constructeur `Function`

Le constructeur `Function` crée un nouvel objet `Function`. En JavaScript, chaque fonction est un objet `Function`.

Appeler ce constructeur permet de créer des fonctions dynamiquement mais cette méthode souffre de défauts équivalents à `eval` en termes de sécurité et de performance. Toutefois, à la différence d' `eval`, le constructeur `Function` permet d'exécuter du code dans la portée globale.

Exemple

```
const calculAire = new Function('a', 'b', 'return a * b');

console.log("Aire : "+calculAire(5, 6)); // Affiche 30
```

L'usage du constructeur `Function` est à proscrire.

Arguments et retours

Passage d'arguments

+++ TODO +++

Le mot-clé `return`

+++ TODO +++

> Une fonction qui ne retourne pas de valeur est appelée une procédure.

Callbacks

+++ TODO +++

Autres points sur les fonctions

Fonctions anonymes

Les fonctions anonymes (appelées aussi "lambdas" dans certains langages) sont des fonctions qui ne portent pas de... nom.

Leur utilité est que le développeur peut les écrire où bon lui semble dans le code quand il en a besoin.

Syntaxe traditionnelle

Syntaxe fléchée

Fonctions auto-invoquées

Les fonctions anonymes peuvent être auto-invoquées, c'est-à-dire qu'elles peuvent s'exécuter seule sans aucun appel ailleurs dans le code, de façon automatique donc.

+++ TODO : utilité et exemple +++

Fonctions imbriquées

+++ TODO +++

Fonctions récursives

+++ TODO +++

Exercices

Exercice 1

Traduire la déclaration de fonction suivante en expression de fonction.

+++ TODO +++

Exercice 2

Ecrire une procédure qui affiche le produit de 2 nombres saisis au clavier.