Université Jendouba

Faculté des Sciences Juridiques Économiques et de Gestion

Développement Web: JavaScipt Avancé

Manel Ben Sassi

Université de Jendouba, FSJEG

bensassi.manel@gmail.com

Sommaire

Les objectifs ...

A la fin de ce cours, vous devriez être capable de ...

- Différencier les différents types de fonction
- Ecrire une fonction en JavaScript de différentes manière
- Définir la portée des variables par rapport à la définition des fonctions.
- Comprendre le fonctionnement de la fonction "*CallBack*" de JavaScript

Les types de fonctions en JS ...

Rappel sur les focntions en JS

Objectifs : programmation modulaire, factoriser et réutiliser le code.
⇒ *Retenez qu'une fonction, ce n'est pas si différent qu'une variable...*

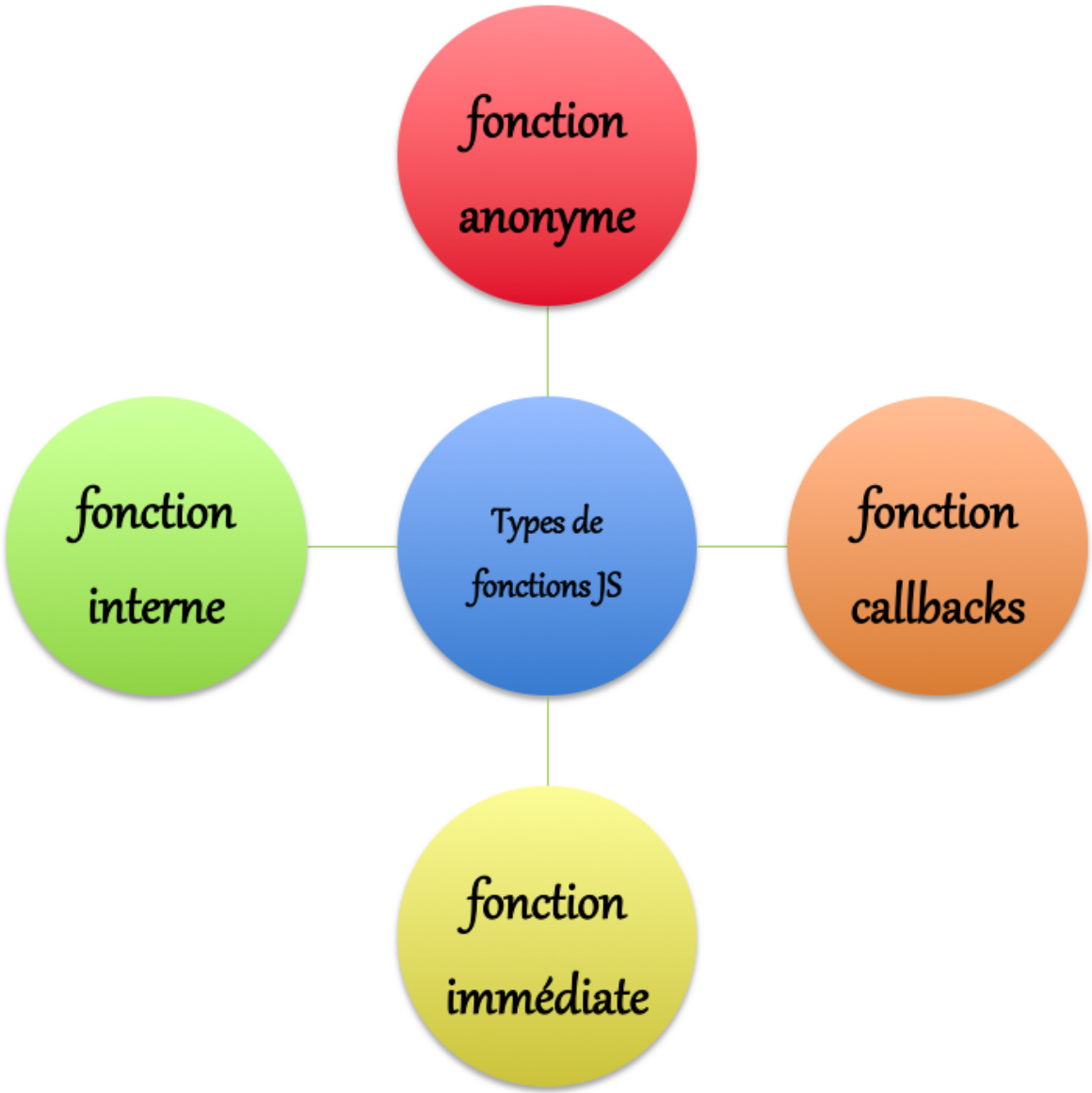
```
1  function sum(a,b){
2      return a+b;
3  }
4  var c = sum(1,2);
5  console.log(c);
```

```
7  var resultat= function(a,b){
8      return (a+b);
9   }
10  console.log(resultat(1,2));
11
```

Figure – Une focntion toute simple

Qu'en pensez-vous d'une démonstration sous JSFiddle ?
https ://jsfiddle.net/

Les fonctions en JS



Les fonctions internes (Inner) ...

Les fonctions internes (Inner)

- Les fonctions sont typées comme des variables
- Nous pouvons les déclarer dans une fonction comme de simples variables.

```
17 (function(){
18     var s="world";
19     console.log("Hello"+s);
20 })();
```

Les fonctions anonymes ...

Les fonctions anonymes

Une fonction anonymes est une fonction qui n'a pas de nom

```
7  // function anonyme
8  function(){
9      alert('Hello');
10 }
```

Très peu utilisée cette manière, nous préférons

```
11 // Function anonyme
12 var a= funtion(){
13     alert('Hello');
14 }
```

Objectifs : isoler son code

```
12 // Code externe
13 function() {
14     // Code isolé
15 };
16 // Code externe
```

Les fonctions immédiates (Self-invoking)

...

Les fonctions immédiates (Self-invoking)

- Les fonctions immédiates sont une application des fonctions anonymes, mais appelées directement après leurs créations

```
// Code externe
(function() {
    // Code isolé
})();
// Code externe
```

```
(function(){
    var s="world";
    console.log("Hello"+s);
})();
```

Figure – L'exécution immédiate de la focntion auto-appelante

Les fonctions Callback ou auto-appelante ...

Les fonctions callback

- Les fonctions sont typées comme des variables ⇒ Nous pouvons les passer en paramètre

```
1  //fonction d'affichage
2  var show = function affichage(c){
3      console.log("somme "+c);
4  }
5  //focntion somme
6  function sum(a,b, affichage){
7  var c = a+b;
8  affichage(c);
9  return c;
10 }
11 // appel de la fonction
12 var c = sum(10,4,show);
13 //afficher la valeur
14 console.log(c);
```

Qu'en pensez-vous d'une démonstration sous JSFiddle ?

https ://jsfiddle.net/

Les fonctions callback

Les fonctions Callback ou "de rappel" par définition ...

- C'une fonction qui est passée en paramètre à une autre fonction.
- Elle est exécutée au "bon moment".
- Elle est souvent utilisée pour **la programmation évenementielle** ou encore la programmation **Asynchrone** (exemple l'AJAX).

Les fonctions callback : Démonstaration

Protocole de démonstration d'utilité !

Lien de la démonstration : *https ://jsfiddle.net/manelbs/enn74tff/2/*

- Objectif : Nous souhaitons que chaque Li ait son propore clique !
- 1ère étape : Nous ajoutons une boucle et un affichage classique
- 2ème étape : la solution avec la fonction auto-appellante

Scope et *Closure* des fonctions ...

Le Scope ou la portée des fonctions

Deux types de portée

```
/*--------
        Global Scope
        ---------------*/
    function test(){
        /*---------
            instruction
            Local Scope
                    -------*/
    }
     /*--------
        Global Scope
        ---------------*/
```

Démonstration Scope sous JSFiddle ?

https ://jsfiddle.net/

Very important

La fermeture des fonctions

Il y a la possibilité d'imbriquer la déclaration des fonctions

```
> function externe(x) {
    function interne(y) {
      return x + y;
    }
    return interne;
  }
  var fn_interne = externe(3);
  var resultat = fn_interne(5); // renvoie 8

  var resultat1 = externe(3)(5); // renvoie 8
```

La fermeture des fonctions

Fermeture des fonctions

Une fermeture est une fonction possédant des variables libres ainsi qu'un environnement qui **ferme** sur ces variables.

Une fonction imbriquée est une **fermeture** $\Rightarrow$ La fonction imbriquée peut **"hériter"** des arguments et des variables de la fonction qui la contient $\Rightarrow$ *La fonction interne contient la portée de la fonction externe*.

Retenez

- On ne peut accéder à la fonction interne seulement avec des instructions contenues dans la fonction externe,
- La fonction interne est une **fermeture** $\Rightarrow$ la fonction interne peut utiliser des arguments et des variables de la fonction externe alors que la fonction externe ne peut pas utiliser de variables et d'arguments de la fonction interne.

Exemple ?

Exemple de fermeture et d'imbrication

La porté des variables de la fonction C par rapport à la fonction A et B ?

```
> function A(x) {
    function B(y) {
      function C(z) {
        console.log(x + y + z);
      }
      C(3);
    }
    B(2);
  }
  A(1); // crée un message d'alerte avec 6 (= 1 + 2 + 3)
```

Exemple de fermeture et d'imbrication

C accède à la variable y de B et à la variable x de A. Cela est possible parce que :

1. B est une fermeture $\in$ A $\Rightarrow$ B peut accéder aux arguments et aux variables de A
2. C est une fermeture $\in$ B
3. Étant donné que la fermeture de B $\in$ A et que celle de C $\in$ B, C peut accéder à la fois aux arguments et variables de B et A. $\Rightarrow$ C enchaîne les portées de B et A dans cet ordre.
4. La réciproque n'est pas vraie. A ne peut avoir accès à C, parce que A ne peut accéder ni aux variables ni aux arguments de B, or C est une variable de B. C est donc privé et seulement pour B.

Pour résumer ...

En conclusion

- Il existe des fonctions natives, mais il est aussi possible d'en créer, avec le mot-clé **function**.
- Les variables déclarées avec **var** au sein d'une fonction ne sont accessibles que dans cette fonction.
- Il faut éviter le plus possible d'avoir recours aux variables globales.
- Une fonction peut recevoir un nombre défini ou indéfini de paramètres. Elle peut aussi retourner une valeur ou ne rien retourner du tout.
- Des fonctions qui ne portent pas de nom sont des fonctions anonymes et servent à isoler une partie du code.

Références Bibliographiques

Références Bibliographiques

1. JavaScript : La référence, **David Flanagan**, 5^{me} Edition O'Reilly, 1042 page, 2007
2. Apprendre à Développer avec JavaScript, **Christian Vigouroux**, Edition ENI, 726 pages, 2016
3. Débuter en JavaScript, **Shelley Powers**, Edition Eyrolles, 2007