JavaScript

NÉCESSAIRE WEB

XAVIER BLANC – UNIVERSITÉ DE BORDEAUX

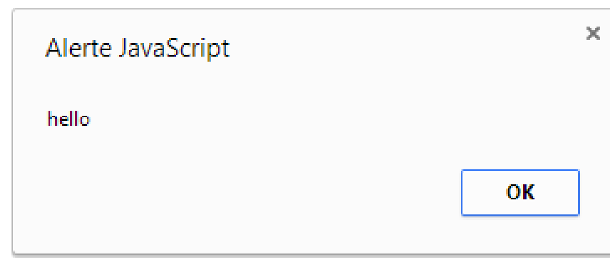
Script

JavaScript est un langage de script

Un script est une séquence d'instructions qui seront interprétées les unes après les autres

Les instructions permettent de

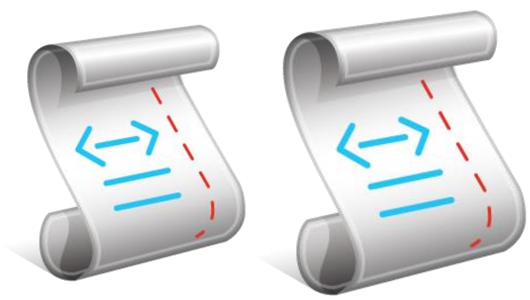
- Interagir : savoir qu'un bouton a été cliqué, qu'une donnée est mise à jours
- Afficher : manipuler le DOM pour rendre visible des nouvelles parties
- Communiquer : envoyer ou recevoir des requêtes (http ou base de données)



Interpreteur – Machine Virtuelle

Les scripts sont interprétés par une machine virtuelle

- Dans le navigateur web (Front)
- *ou sur le serveur (Back)*



Interpreteur – Mémoire

L'interpreteur dispose d'une mémoire globale (**Objet global**)

Cette mémoire possède les éléments définis par les scripts (variables, objets, fonctions, etc.)

Si l'interpreteur charge deux scripts, ces scripts partagent le même objet global et donc les mêmes variables, objets et fonctions !

Variables et Types

Toutes les variables sont typées dynamiquement dans JavaScript

Les types de bases sont:

- Chaîne de caractères, nombre, booléen

Les types complexes sont:

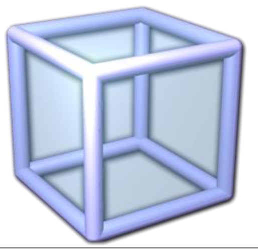
- Objet, Tableau, Fonction, …

Toute variable peut être interprétée comme un boolean

- If (!0) …

typeof permet de connaître le type d’une variable

Objet



En JavaScript tout est objet (… mais objet JavaScript)

Un objet est un ensemble de propriétés (nom, valeur)

Il est possible de modifier les propriétés d’un objet à n’importe quel moment dans un script

test.js | index.html | test.htm

```
1  monMooc = {};
2  monMooc.titre = "JavaScript pour MEAN";
3  monMooc.nbView = 5;
4  alert(monMooc.nbView);
5  monMooc.nbView = 10000000;
6  alert(monMooc.nbView);
```

test.js

```
1  monMooc = {
2      titre: "JavaSc
3      nbView: 5
4  };
5  alert(monMooc.nbVi
6  monMooc.nbView = 10000000;
7  alert(monMooc.nbView);
8
```

Alerte JavaScript

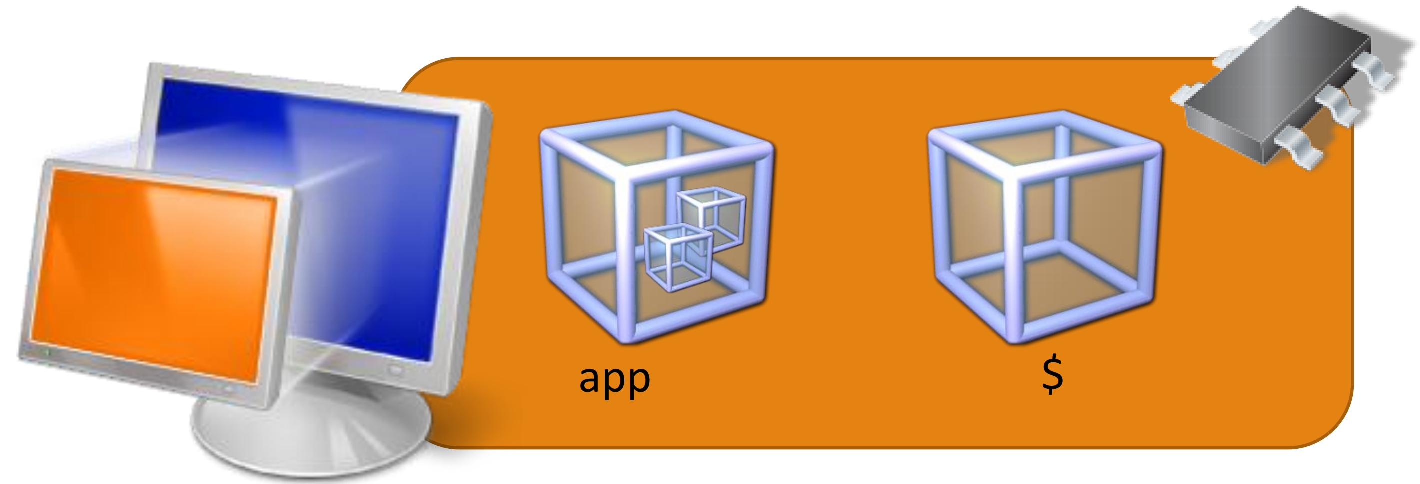
10000000

☐ Empêcher cette page de générer des boîtes de dialogue supplémentaires

OK

Objet et espace de nom

Les objets sont souvent utilisés comme espace de nom pour organiser la mémoire



Fonction

Une fonction est un objet JavaScript

Une fonction a un nom, des paramètres (typage dynamique) et du code

Une fonction peut être définie:

- (global) dans un script, (inner) dans une fonction ou (méthode) dans une propriété d'un objet

Une fonction peut définir des variables propres (mot clé **var** !)

Une fonction retourne toujours quelque chose (undefined si non spécifié)

Les fonctions (global, inner, ou méthode) ont une propriété **this** qui dépend de la façon dont l'appel sera effectué

Inner Fonction et Closure

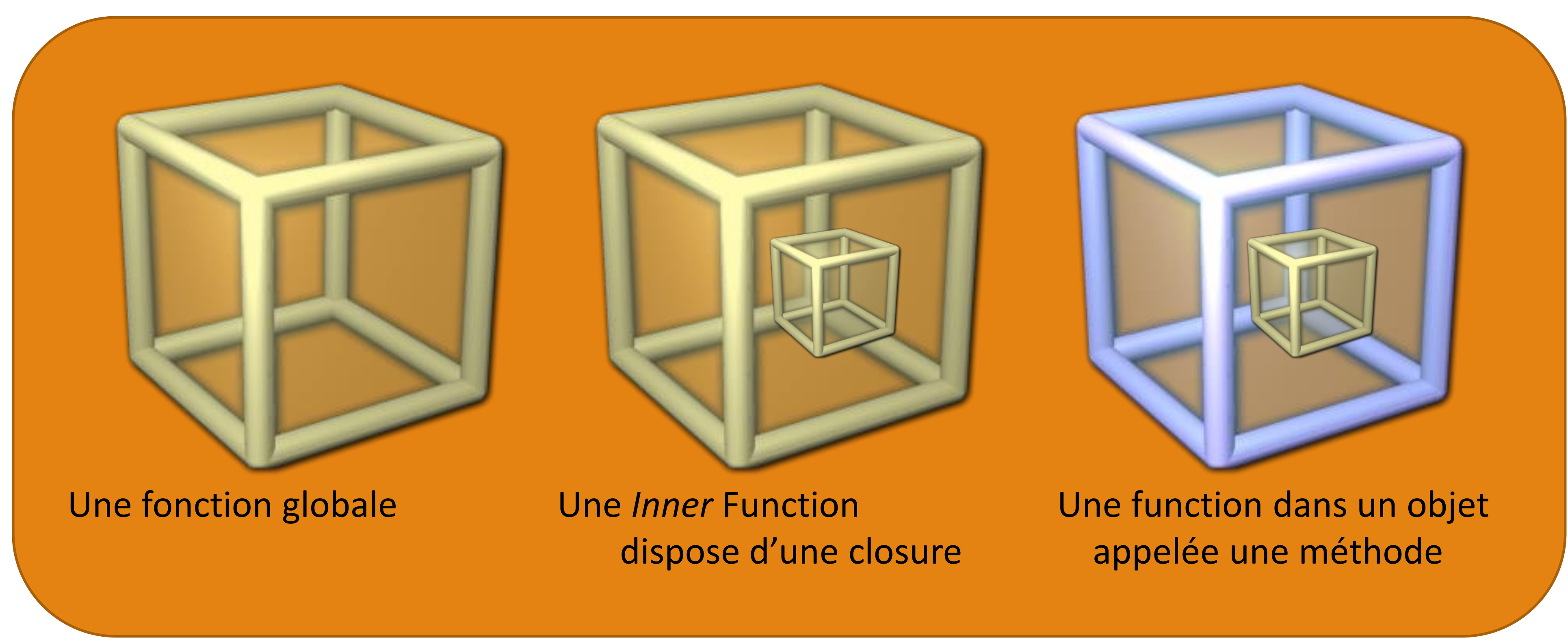
Une inner fonction a toujours accès aux variables propres et aux paramètres de la outer fonction qui l'a définie (**closure**)

Cet accès reste valable quand la inner fonction sera appelée. Même si la outer fonction s'est terminée.

Cela permet de masquer des variables
qui ne seront accessibles que par la inner fonction

```
22  function outer () {
23      var v=0;
24
25      function inner() {
26          v++;
27          return v;
28      }
29      console.log(v);
30
31      return inner;
32  }
33
34  f = outer();
35  console.log(f());
36  console.log(outer.v);
37  console.log(f());
```

Fonction



Fonction et appel de fonction

Une fonction peut être appelée n'importe où dans le scope où elle a été définie (avant ou après sa définition)

Si l'interpreteur appelle la fonction, il s'arrête momentanément pour exécuter le code de la fonction, puis une fois la fonction terminée il reprend là où il en était

```
test.js
1  function say(s) {
2      alert(s);
3  };
4
5  say("hello");
6
```

Appel de fonction et **this**

Lors d'un appel d'une global fonction ou d'une inner fonction, **this est toujours lié à la mémoire globale**

```
nbCall = 0;
function say(s) {
    trace();
    alert(s);

    function trace() {
        this.nbCall++;
    }
};

say("hello");
alert(nbCall);
```

Pour appeler une méthode, il faut passer par l'objet, **this est alors lié à l'objet**

```
o =  {
    nom: "blanc",
    prenom: "xavier",

    setNom : function (n) {
        this.nom = n;
    }
};

alert(o.nom);
o.setNom("inconnu");
alert(o.nom);
```

Fonction et apply

Une fonction est un objet

Qui a une méthode nommée apply

Cette méthode permet d'appeler la fonction
en passant comme paramètre **this**
et les **paramètres** de la fonction dans un tableau !

```
test.js
1  v = 0;
2  function setV(nV) {
3      this.v = nV;
4  }
5  o = { v: 1};
6  setV.apply(o,[3]);
7  alert(v);
8  alert(o.v);
9
```

Fonction et new

Une fonction peut être appelée avec **new** (on dit qu'on exécute le constructeur)

Dans ce cas, (1) un nouvel objet est créé, et (2) la fonction est appelée avec cet objet lié à **this**

L'objet créé à un lien vers la fonction qui l'a créé (son constructeur)

Cela permet de définir des objets similaires (!=classe)

test.js

```
function setV(nV) {
    this.v = nV;
};
o = new setV(3);
alert(o.v);
```

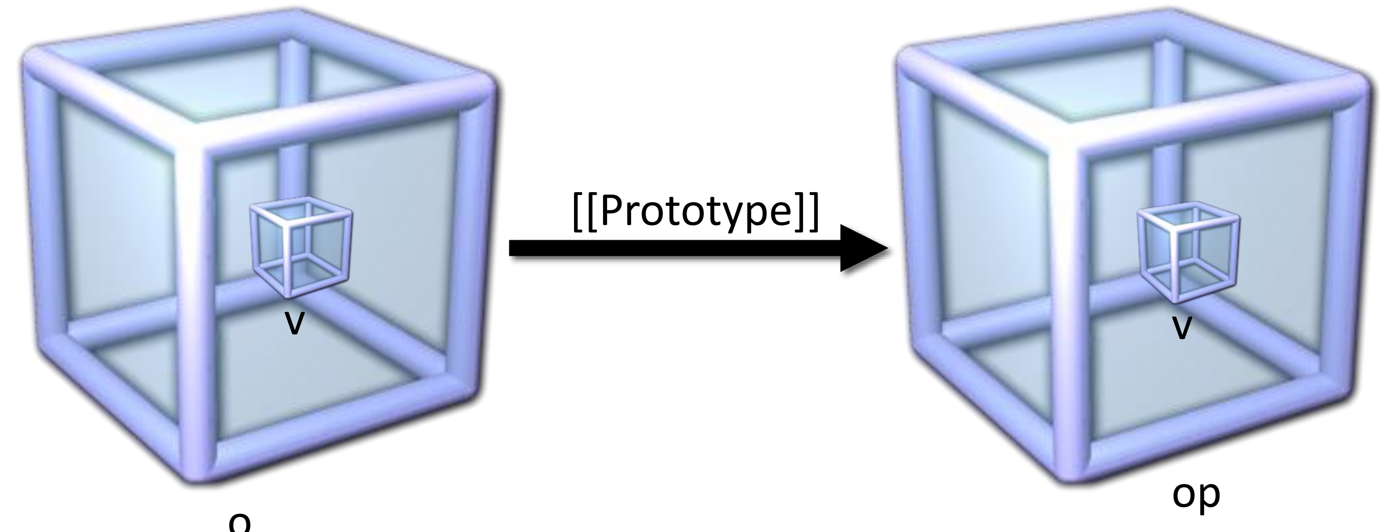
Prototype (!= héritage)

Tout objet est lié a un prototype (qui est un objet)

Si un objet n’a pas une propriété, il demande à son prototype, etc.

Si on met à jours une propriété définie dans un prototype, on l’ajoute à l’objet

```
o.v;     //[[proto]] op.v
o.v++; //ajout
o.v;    //o.v
```



Chaines de prototypes

o={}

- le prototype de o est Object.prototype

t=[]

- le prototype de t est Array.prototype qui a Object.prototype comme prototype

function f(){}

- le prototype de f est Function.prototype qui a Object.prototype comme prototype

o= new f();

- le prototype de o est f.prototype quand f est exécuté

o2=Object.create(o);

- le prototype de o2 est o

Fonction, méthode et objet => OO

Fonction pour définir une fabrique (constructeur)

Exploitation des variables pour les propriétés private

Utilisation du prototype pour les champs statique

```
30  function Person() {
31      var first = "";
32      var last ="";
33      this.getFirst = function() {return first;};
34      this.setFirst = function(v) {first = v;};
35      this.getLast = function() {return last;};
36      this.setLast = function(v) {last = v;};
37
38      Person.prototype.nbInst++;
39  }
40
41  Person.prototype.nbInst = 0;
42
43  me = new Person();
44  alert(me.getFirst());
45  me.setFirst("Xavier");
46  alert(me.getFirst());
47  alert(Person.prototype.nbInst);
48  him = new Person();
49  alert(Person.prototype.nbInst);
50
```

Fonction Paramètre

Un paramètre d'une fonction peut être une fonction !

La fonction passée en paramètre peut être appelée dans la fonction principale (quel mode d'appel?)

Il est possible de définir la fonction au moment de l'appel main(function() {…})

```
test.js
1  function fini() {
2      alert("terminé")
3  };
4
5  function main(f) {
6      //lots of stuff
7      f();
8  };
9
10 main(fini);
```

Fonction Paramètre et CallBack

Imaginons, la fonction *traitement* dans une lib
qui manipule une donnée data
Une partie des traitements est réalisée par une fonction f passée en paramètre.

```
1  function traitement(f) {
2      var data
3      //du code
4      f(data)
5      //du code
6  }
```

Si f est asynchrone, il faut lui passer une autre fonction en paramètre (la CallBack)
qui est définie par *traitement*
et qui sera appelée par f quand sont traitement sera terminé
Par convention, la CallBack a 2 paramètres
err (si erreur) et res (pour le résultat)

```
1  function traitement(f) {
2      var data
3      //du code
4      f(data, cb)
5      var cb=function(err, res) {
6          //code quand f a terminée
7      }
8      //du code
9  }
```

Souvent on définie f de façon anonyme lors de l'appel

```
12  traitement(function(data,cb) {
13      //du code long
14      cb(null, "OK") //pas d'erreur, res="OK"
15  })
```

Array

Les tableau sont supportés nativement par Javascript

Un tableau est un objet qui stocke plusieurs valeurs à des indexes (qui commence par 0)

Un tableau a une propriété length (nb de cases prévues)

Les tableaux sont souvent utilisés pour passer des ensembles de paramètres à des fonctions

test.js

```
a = [1, "2" , "trois" , {v:"quatre"}];
alert(a[1]);
alert(a[3].v);
a[10] = "loin";
alert(a[9]);
```

DOM une API JavaScript

Tout navigateur permet une manipulation du DOM en JavaScript

window variable globale qui représente la fenêtre (page, iframe)

- **location**
- **onload**

document variable globale racine du DOM

- **getElementById()**
- **getElementsByTagName()**
- **createElement()**

element tout nœud du DOM est un élément

- **innerHTML**
- ***onclick***

Exemple JavaScript dans HTML

```html
<!doctype html>
<html lang="fr">
    <head>
        <title>Exemple</title>
        <meta charset="utf-8">
        <script src="js/test.js"></script>
    </head>
    <body>Yo</body>
</html>
```

```javascript
window.onload = ajouterHTML;

function ajouterHTML() {
    document.body.innerHTML = "Ajout de mon code HTML";
}
```