


# Programmation WEB

## Javascript/DOM

---

Denis Monnerat

monnerat@u-pec.fr 

Updated: 2018/01/07

IUT de Fontainebleau



Première partie I

Bases de javascript

# Sommaire

1. Introduction
2. Survol du langage
3. Javascript et html

# Introduction

---

Une application Web est une applications clients/serveur(s)



On peut la voir en trois couches ...



Front-End

Back-End

SGBD



PostgreSQL



ORACLE®



SGBD



Back-End





# Front-End

```
graph TD; FE[Front-End] --> S[Structure]; FE --> P[Présentation]; FE --> A[Applicatif];
```

Structure

Présentation

Applicatif

# Front-End



Présentation

Applicatif

# Front-End



↓

Applicatif

# Front-End



Javascript est même utilisé coté serveur (Back-end) avec




- Les bases de JS reposent sur ECMAScript





spécifié et standardisé par ECMA ("European Computer Manufacturers Association").

- Dernière version : ECMA-262 Edition 6 (17 juin 2015)

<http://www.ecma-international.org/ecma-262/6.0/index.html> 

Très bonnes documentations ici

<https://developer.mozilla.org/fr/docs/Web/JavaScript> 

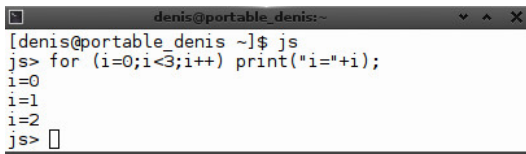
<http://www.w3schools.com/> 

Autant de moteurs javascript qu'il y a de navigateurs :

- SpiderMonkey pour firefox (écrit en C).
- JavaScriptCore pour safari (écrit en C++).
- V8 JavaScript engine pour chrome et chromium (écrit en C++).
- Carakan pour Opéra.
- Chakra pour internet explorer.



## Moteur spiderMonkey (mode console)



```
denis@portable_denis: ~  
[denis@portable_denis ~]$ js  
js> for (i=0;i<3;i++) print("i="+i);  
i=0  
i=1  
i=2  
js> █
```



# Javascript

Domaine d'application : initialement le web ...

- Interface graphique des applications web.
- Validation des données.
- Modification dynamique de l'arbre du document. (DOM)
- googlemaps

Mais Javascript est aussi le langage de développement sur certains OS où les applications sont des applications HTML5 au sens large.

TIZEN™ 

 Firefox OS

## Beaucoup d'apis ....

- Ajax (XMLHttpRequest) : requête http asynchrone.
- WebWorkers : programmation parallèle.
- WebSockets : communication bidirectionnelle entre le client et le serveur.
- Canvas 2D Context : dessin avec la balise `canvas`.

## ... de bibliothèques et framework

- JQuery
- Mootools, Angular.js, Prototype.js, Dojo, React, Ember, Backbone, etc.
- Pixi, Phaser (jeux)

# Interaction avec html

Il existe **deux façons** d'interagir avec une page html à partir de javascript :

## DHTML (historique)

- Nouveaux attributs pour la gestion événementielle (onClick, etc ....)
- Accès aux éléments par les objets prédéfinis (document.forms[0].elements[1])

## DOM

- La page est un arbre. On peut accéder aux noeuds, les modifier, etc ...
- Gestionnaire d'événements intégré à DOM.
- Modèle générique.

# Survol du langage

---

# Le langage javascript

C'est un langage à objets utilisant le concept de prototype, disposant d'un typage faible et dynamique qui permet de programmer suivant plusieurs paradigmes de programmation : fonctionnelle, impérative et orientée objet.

Javascript permet, entre autres :

- Programmation événementielle. ex : changer une image au survol du pointeur.
- Effectuer des calculs. ex : lire la valeur saisie, la multiplier par 3.14 et afficher le résultat.
- Modifier la présentation et/ou le contenu du document html.
- Effectuer des requêtes HTTP.
- Accéder à des bases de données locales, etc.

Au niveau du langage, on distingue :

### Le noyau, qui comporte

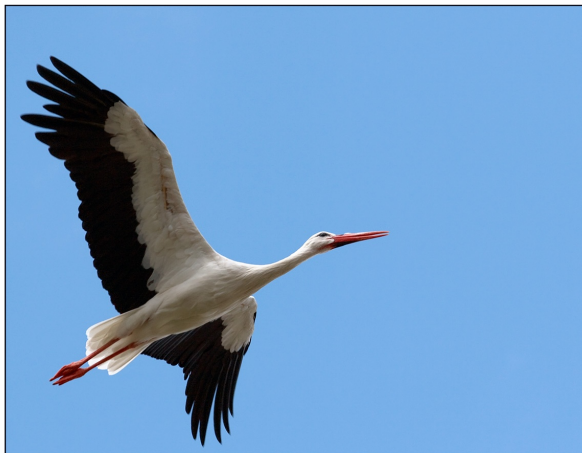
- des objets prédéfinis (tableaux, etc...),
- des opérateurs,
- des structures algorithmiques.

### La couche navigateur

un ensemble d'objets associés au navigateur.

- fenêtres,
- formulaires,
- images, etc.

La suite n'est aucunement un cours complet sur Javascript, à peine un survol avec beaucoup d'oublis (volontaires).



# La base

- Opérateurs, boucles, conditionnelles similaires au C.
- **Mais**, variables typées dynamiquement
- **;** comme en C marque la fin d'une instruction. (optionnel)
- Mot clé **var** pour déclarer une variable. (optionnel)

```
for (var count=0;count<5,count++)
{
    print("valeur="+count);
}
var count=0;
var fini=false;
while(!fini)
{
    if (count>4)
    {
        fini=true;
    }
    else
    {
        print("valeur="+count);
    }
    count++;
}

var count=0;
do
{
    print("valeur="+count);
}while(count<5);
```



# Types simples

- Booléens (Boolean).
- Nombres (Number).
- Valeurs prédéfinies
  - true
  - false
  - undefined
  - Infinity
  - NaN

```
js> var x=2;
js> var y=3;
js> var z=x+y;
js> z;
5
js> z==(x+y);
true
js> (x==y);
false
```

```
js> z++;
5
js> ++z;
7
js> x/3;
0.6666666666666666
js> x>y? 0:1;
1
js> 128<<1;
256
js> 255|128;
255
js> 255&128;
128
js> 0x23;
35
js> 034;
28
js> 5=="5";
true
js> 5==="5";
false
js>
```

# Arrays (les tableaux)

- Numérique ou associatif.
- Éléments de type hétérogène.
- Syntaxe : `a[1]=1` pour `[0,1,true]`.
- Propriété `length` pour la taille : `a.length` vaut 3.
- N'existe pas pour les tableaux associatifs.
- Beaucoup de méthodes (notamment `forEach`).

```
js> var tab=[1,2,3];
js> var tab2=tab;
js> tab.push(5);
4
js> tab;
[1, 2, 3, 5]
js> tab.pop();
5
js> tab.unshift(5);
4
js> tab;
[5, 1, 2, 3]
js> tab.reverse();
[3, 2, 1, 5]
js> tab2;
[3, 2, 1, 5]
js> tab.sort();
[1, 2, 3, 5]
js>tab.splice(1,2,6,7);
[2, 3]
js>tab;
[1, 6, 7, 5]
js>tab.slice(1,3);
[6,7]
```

- Type String.
  - Beaucoup de méthodes.
  - Littéraux : 'val' ou "val".
- Expressions régulières
  - Support des expressions régulières classiques (grep unix).
  - littéraux : /pattern/flags.

```
js> 'wim web'.indexOf('i');
1
js> 'wim web'.lastIndexOf('w');
4
js> 'wim web'+ ' semestre1';
"wim web semestre1"
js> 'wim web'.split(" ");
["wim", "web"]
js> 'wim web'.substr(1,4);
"im w"
js> 'wim web'.slice(1,4);
"im "
js> 'wim web'.toLocaleUpperCase();
"WIM WEB"
js>'wim web'[2];
"m"
js>/~w.+/i.test("wim web");
true
js> "j'aime bien javascrit".match(/\S+/g);
["j'aime", "bien", "javascrit"]
js>
```

# Les objets

- Comme les structures en C : regroupent des couples (clé,valeur).
- peuvent être vus comme des tableaux indicés.
- utilisés pour les tableaux associatifs.
- Syntaxe dédiée : JSON.

```
js> var obj={un:1,deux:2,trois:3,verite:true};
js> obj.deux;
2
js> obj['deux']
2
js> var obj2=obj;
js> obj.newprop="coucou";
"coucou"
js> obj2;
({un:1, deux:2, trois:3, verite:true,
  newprop:"coucou"})
js> obj2;
({un:1, deux:2, trois:3, verite:true,
  newprop:"coucou"})
js> obj['tab']=[true,1];
[true, 1]
js> obj
({un:1, deux:2, trois:3, verite:true,
  newprop:"coucou", tab:[true, 1]})
js> "trois" in obj;
true
js>obj.tab[0];
true
js>obj.tab[1];
```

# Les Dates

- js fournit une "classe" prédéfinie Date.
- timestamp.
- Plusieurs constructeurs.
- Beaucoup de méthodes.

```
> var maintenant=new Date();
js> maintenant;
(new Date(1386414882694))
js> maintenant.toString()
"Sat Dec 07 2013 12:14:42 GMT+0100 (CET)"
js> var demain=new Date(2013,12,8,10,15,10)
(new Date(1389172510000))
js> demain.toString()
"Wed Jan 08 2014 10:15:10 GMT+0100 (CET)"
> demain.getDate();
8
js> demain.getDay();
0
js> demain.getMonth();
11
js> demain.getFullYear();
2013
js> demain.getMinutes();
15
js> demain.toDateString();
"Sun Dec 08 2013"
```

# L'objet Math

- L'objet global Math fournit beaucoup de fonctions et constantes mathématiques.
- `Math.abs`, `Math.ceil`, `Math.round`, `Math.log`, `Math.sin`, `Math.sqrt`, `Math.random`, etc.
- `Math.PI`, `Math.E`, `Math.SQRT2`, etc.

```
js> Math.PI;
3.141592653589793
js> Math.E;
2.718281828459045
js> Math.sin(Math.PI/2);
1
js> var rnd=Math.random();
js> rnd;
0.5382747752890101
js> Math.pow(Math.cos(rnd),2)+
  Math.pow(Math.sin(rnd),2);
0.9999999999999999
js> Math.max(10,2,3,-5,100,8)
100
js> Math.sqrt(64)
8
js> Math.log(2) == Math.LN2;
true
```

# Les fonctions

- `function` est un type du langage. (une "classe")
- On peut passer des fonctions en arguments d'autres fonctions.
- Des fonctions peuvent renvoyer des fonctions.
- Fonctions anonymes.
- Portée d'une variable définie uniquement par les blocks fonctionnelles.
- Des fonctions peuvent être des membres d'objets.

```
js>function f(a) {return 2 + a;};
js>typeof f;
"function"
js>f(3);
5
js>f("toto");
"2toto"
js>e=function somme(a,b){return a+b;};
js>e(3,6);
9
js>function fun(a,op,b)
  {
    return op(a,b);
  };
js>fun(10,somme,30);
40
js>fun(10,function(x,y){return x-y;},20);
js>-10
js>(function(x){return x+2;})(3)
5
```

# Javascript et html

---



# A quoi ça sert ?

Javascript est devenu le langage de script, interprété par tous les navigateurs, qui permet entre autres :

- Modifier dynamiquement la présentation (le css) de la page.
- Modifier dynamiquement la structure du document html.
- Réagir aux événements utilisateurs.
- etc

Un sur-ensemble du langage est disponible, avec des objets prédéfinis spécifiques au web :

- fenêtres,
- écran,
- formulaires,
- images, etc.

# Exécution de code

On peut (faire) exécuter du code javascript de plusieurs manières.



1. Avec la balise `<script>`.
  - déclarations dans l'en-tête (`<head>...</head>`), souvent, mais possibilité d'insertion dans le body.
  - appel de fonction ou exécution de javascript dans le corps (`<body>...</body>`).
  - insertion de fichiers javascript "externes".
2. Clic sur un lien : `<a href="javascript:...">ici</a>`.
3. En réponse à un événement sur un élément html : attribut `onEvenement`. Le code javascript est exécuté lors de l'événement correspondant (DHTML).

# Inclusion : dans le code html

```
<html>
  <head>
    <script language="javascript">
      function fin(){
        alert('Bye');
      }
    </script>
  </head>
  <body>
    <script language="javascript">
      document.write('Pour se dire aurevoir');
      /* inutile de passer par js (pourquoi ?)*/
    </script>
    <br /><a href="javascript:fin();">cliquez ici</a>
    <br />ou passez la souris sur
    <a href="" onMouseOver="fin();">ce lien</a>
  </body>
```

# Inclusion : depuis des fichiers externes

```
<html>
  <head>
    <script language="javascript" src="fin.js"></script>
  </head>
  <body>
    Pour se dire aurevoir
    <br /><a href="javascript:fin();">cliquez ici</a>
    <br />ou passez la souris sur
    <a href="" onMouseOver="fin();">ce lien</a>
  </body>
</html>
```

```
function fin(){ /* fichier fin.js */
  alert('Bye');
}
```

# Objets prédéfinis

Objets instanciés au démarrage du navigateur. Ils permettent d'accéder à des informations concernant le navigateur, les documents html affichés, l'écran de la machine.

## ○ Navigator

- une seule instance, l'objet `navigator`.
- infos sur le nom, version, plugins installés.

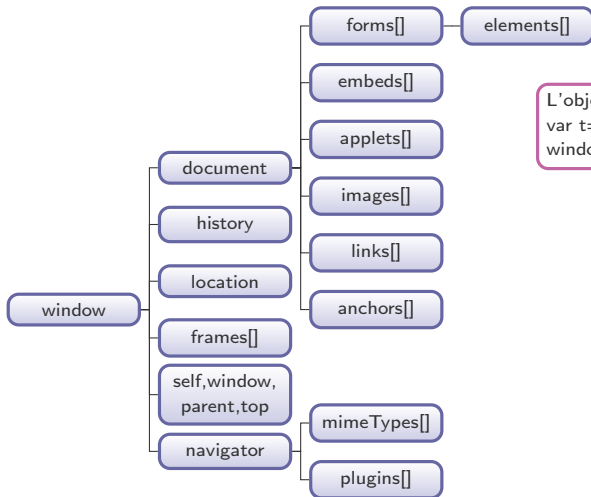
## ○ Window

- une instance par fenêtre et frame du document html.
- accès à tous les objets créés par les balises html.

## ○ Screen

- une instance, l'objet `screen`
- infos sur la largeur et hauteur en pixels, couleurs disponibles, etc...

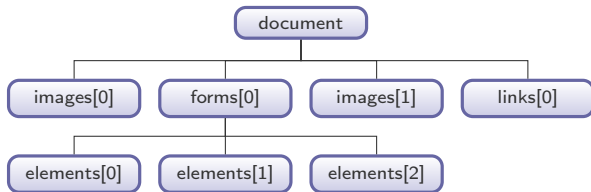
# Hiérarchie des objets du navigateur



L'objet `window` est la racine.  
`var t=10 ; // variable nouvelle`  
`window.t=10 ; //idem`

# Accès aux éléments du document html (version dhtml)

Un champ de saisie est contenu dans un formulaire contenu dans le document



```
<script>
  document.forms.formulaire.
    elements.adresse.value="???";
  document.forms[0].elements[0].value="???";
</script>
<form name="formulaire">
  <input type="texte" name="adresse">
</form>
```

On peut accéder à un élément d'un de ces tableaux avec l'attribut `name` de la balise html.



Pour les plus hardis, il est possible de récupérer n'importe quel noeud (node) du document html. En particulier, si cet élément est identifié (attribut id). Cela renvoie à la représentation DOM du document.

```
<div id="container">  
  <h1>ceci est une division</h1>  
  <p>et cela un paragraphe</p>  
</div>
```

```
var mondiv=document.getElementById("container");  
mondiv.style.backgroundColor="#fea360";
```



# Sensibilisation

```
onEvenement="Action_Javascript_ou_Fonction();
```

ou en assignant la propriété correspondante de l'élément

```
elm.onclick=function;
```

```
<!DOCTYPE html>  
<html>  
  <body>  
    <button type="button" onclick="alert('Hello world!')">  
      Click Me!  
    </button>  
  </body>  
</html>
```

---

<code>onAbort</code>	<b>en cas d'interruption</b>
<code>onBlur</code>	<b>en quittant</b>
<code>onChange</code>	<b>après modification réussie</b>
<code>onClick</code>	<b>en cliquant</b>
<code>onDbClick</code>	<b>en double-cliquant</b>
<code>onError</code>	<b>en cas d'erreur</b>
<code>onFocus</code>	<b>en activant</b>
<code>onKeyDown</code>	<b>en appuyant sur une touche</b>
<code>onKeyPress</code>	<b>en maintenant une touche appuyée</b>
<code>onKeyUp</code>	<b>en relâchant la touche</b>
<code>onLoad</code>	<b>en chargeant le document</b>

---

---

<code>onMouseDown</code>	<b>en maintenant la touche de souris appuyée</b>
<code>onMouseMove</code>	<b>en bougeant la souris</b>
<code>onMouseout</code>	<b>en quittant l'élément avec la souris</b>
<code>onMouseover</code>	<b>en passant sur l'élément avec la souris</b>
<code>onMouseUp</code>	<b>en relâchant la touche de souris</b>
<code>onReset</code>	<b>en initialisant le formulaire</b>
<code>onSelect</code>	<b>en sélectionnant du texte</b>
<code>onSubmit</code>	<b>en envoyant le formulaire</b>
<code>onUnload</code>	<b>en quittant le fichier</b>
<code>javascript:</code>	<b>pour les liens</b>

---

- Consulter la doc pour la sensibilité des différentes balises.
- Pour la plupart des événements, le navigateur possède déjà un comportement par défaut.
- Quand un événement est intercepté, le navigateur exécute d'abord le traitement associé par le gestionnaire, puis celui par défaut.
- Pour empêcher le traitement par défaut, il faut retourner `false`.

```
<input type="text" value="" onKeyPress="return false;">
```



Pour être certain d'exécuter du javascript après que **tout** le document html est prêt :

- utiliser l'événement `onLoad` du `body`.
- placer le code dans une balise `script` à la fin du fichier `html`.

# Sensibilisation dans un attribut html

Le code exécuté en réponse à un événement s'appelle un callback, où une fonction réflexe. Il est possible de transmettre des paramètres à cette fonction depuis l'html.

En particulier :

- l'élément html qui en est le siège avec le mot clé `this`
- l'objet `event` qui représente l'événement lui-même, et qui regroupe des renseignements contextuels : coordonnées de la souris, touche pressée, etc ...
- on peut mettre ce que l'on veut.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      function ok(t,e,m){
        alert(t+": "+e+": "+m);
      }
    </script>
  </head>
  <body>
    <button type="button"
      onclick="ok(this,event,'ok')">
      Click Me!
    </button>
  </body>
</html>
```

Quelques propriétés de l'objet event (il y a différents types d'événements : clavier, souris, etc ...) :

- type : type.
- target ou srcElement : siège.
- which ou keyCode : code de la touche.
- which ou button : bouton souris.
- screenX|Y,clientX|Y,x|y,pageX|Y,layerX|Y,offsetX|Y : position souris.

```
function toto(el,ev)
{
  alert("element "+el);
  alert("touche pressee "+String.fromCharCode(e.which));
}

<input type="text" value="" onKeyPress="clavier(this,event);">
```

# Sensibilisation avec le DOM HTML

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <button id='mybtn' type="button">
      Click Me!
    </button>
    <p id="date"></p>
  </body>
</html>
<script>
document.getElementById('toto').onclick=function(){
  document.getElementById('date').innerHTML=Date();
}
</script>
```

**Attention** la fonction réflexe ne peut recevoir qu'un argument : l'événement lui-même.



# Modification CSS

Chaque objet possède les attributs :

- className

```
document.forms.monformulaire.elements.nom.className="red";
```

- L'attribut style

```
el.style.backgroundColor = "#ff0";  
el.style.color = "#0f0";  
el.style.fontSize = "1.2em";  
el.style.textAlign = "center";
```

Remarque : lorsque la propriété css comporte dans son nom un tiret (background-color), celui-ci est supprimé en javascript, et la lettre d'après est en majuscule.

Deuxième partie II

Javascript "avancé"

4. Les types

5. Objets et fonctions

6. Fonction comme objet

7. Fonctions réflexes et paramètres

# Les types

---

# Typage

Le type est déterminé lors de l'initialisation ou d'une affectation. Il peut changer lors d'une affectation.

## Types primitifs

<code>null</code>	littéral <code>null</code> : représente la "nullité" (aucune valeur pour l'objet n'est présente). type <code>object</code>
<code>undefined</code>	propriété de l'objet global, qui vaut <code>undefined</code> . type <code>"undefined"</code>
<code>boolean</code>	booléens : <code>true</code> , <code>false</code>
<code>number</code>	<ul style="list-style-type: none"><li>○ Entier : <code>102</code>, <code>0xaeF</code>, <code>075</code>.</li><li>○ Réel : <code>3.1415</code>, <code>-6.23e-12</code>.</li></ul>
<code>string</code>	Chaîne de caractères. <code>"toto"</code> , <code>'toto'</code> . Les caractères d'échappement du C sont reconnus

Tout le reste est constitué d'objets et de fonctions (objet aussi).

- ~> `typeof(x)` retourne, sous forme d'une chaîne, le type de `x`. `typeof` sur une variable non définie renvoie la chaîne "undefined".
- ~> A noter la présence de l'opérateur `a===b` qui renvoie `true` si `a` et `b` sont de même type et de même valeur (et `!==`).

Remarques :

- l'opérateur `typeof` ne renvoie pas d'erreur si l'objet n'est pas défini.
- toute variable définie, non initialisé a pour type "undefined".

# Conversion de type

Le type `String` est dominant.

- ~> Conversion implicite avec les opérateurs d'égalités faibles (`==`).
- ~> Pas de conversion avec les opérateurs d'égalités strictes (`===`).

```
N=12;  
T="34";  
X=N+T; // X est la chaîne 1234  
X=N+Number(T); // X vaut 46
```

# null, undefined

```
js> var x=null;
js> typeof(x)
  "object"
js> var y
js> typeof(y)
  "undefined"
js> x==y
  true
js> x===y
  false
js> !x
  true
js> !y
  true
js> z == null
typein:1: ReferenceError: z is not defined
js> z == undefined
typein:2: ReferenceError: z is not defined
js> typeof z
  "undefined"
```

```
js> var s=""
js> s==null
  false
js> !s
  true
js> var t={}
js> t==null
  false
js> !t
  false
js> typeof(t)
  "object"
```



# Objets et fonctions

---

# Les objets

Ils sont traités en interne comme des tableaux associatifs.

- Pas des vraies classes
  - pas de "vrai" héritage.
  - uniquement des créations d'objets et de propriétés prototypes.
  - les méthodes statiques existent.
  - notation pointée.
- Déclaration d'**un** objet avec la syntaxe JSON :

```
var obj=  
{  
  x:2,  
  y:3,  
  somme:function()  
  {  
    return this.x+this.y;  
  }  
};  
alert(obj.x);  
alert(obj['x']);  
alert(obj.somme());
```

- Déclaration d'une classe par la définition de son constructeur.

## Fonction comme objet

---

## Les fonctions sont des objets !

```
js> var obj = {};  
js> var fn = function(){};  
js> obj.prop = "some value";  
js> fn.prop = "some value";  
js> obj.prop == fn.prop  
true
```

Pas de problème. Les fonctions ont des propriétés.

## Une fonction peut

- être affectée à des variables ou des structures de données.
- être passée comme paramètre.
- être retournée par une fonction.
- être constructible lors de l'exécution.

## Utilisation d'un cache :

```
function isPrime( num ) {
  if ( isPrime.cache[ num ] != null )
    return isPrime.cache[ num ];
  var prime = num != 1; // Everything but 1 can be prime
  for ( var i = 2; i < num; i++ ) {
    if ( num % i == 0 ) {
      prime = false;
      break;
    }
  }
  isPrime.cache[ num ] = prime
  return prime;
}
isPrime.cache = {};
js> isPrime(5)
true
js> isPrime.cache[5]
true
```

# Contexte et this

Une fonction s'exécute dans un "contexte" (l'objet duquel la fonction est une propriété) que l'on peut accéder par le mot clé `this` :

ici l'objet global

```
js> this
({})
js> var x=3;
js> this
({x:3})
js> function f(){this.y=4;}
js> f()
js> y
4
js> this
({x:3, f:function f() {this.y = 4;}, y:4})
```

ici l'objet katana

```
js> var katana = {
  isSharp: true,
  use: function(){
    this.isSharp = !this.isSharp;
  }
};
js> katana.use();
js> katana.isSharp
false
```

On peut le changer avec apply ou call

```
js> function S(a){return this.x + a;}  
js> x=2  
js> S.call(this,2)  
4  
js> var obj={x:3}  
js> S.apply(obj,[2])  
5
```

## Tableau arguments :

```
function test() {  
  alert("Nombre de parametres: " + arguments.length);  
  for(var i=0; i<arguments.length; i++) {  
    alert("Parametre " + i + ": " + arguments[i]);  
  }  
}  
test("valeur1", "valeur2");  
test("valeur1", "valeur2", "valeur3", "valeur4");
```



# Le mot clé new

```
function user(prenom,nom){
  this.prenom = prenom;
  this.nom=nom;
  this.changerNom = function (n){
    this.nom=n;
  };
}

js> var Denis = user("denis","monnerat");
js> Denis.prenom
TypeError: Denis is undefined
js> var Moi = new user("denis","monnerat");
js> Moi.prenom
Denis
```

L'opérateur new, suivi de la fonction équivaut à :

```
function user(prenom,nom){
  this.prenom = prenom;
  this.nom=nom;
  this.changerNom = function (n){
    this.nom=n;
  };
}

js> var Denis={};
js> user.call(Denis,"Denis","Monnerat");
```

- On peut voir cela comme la définition d'une classe user, et d'une instantiation avec new.
- Chaque objet garde une trace du "constructeur" avec la propriété (fonction) constructor.

# Closures

```
function f(i){  
  return function (x){  
    return x+i;  
  }  
}  
  
var ADD5 = f(5);  
var x=ADD5(1); // x vaut 6  
var ADD10 = f(10);  
x=ADD10(1); // x vaut 11
```

Très utilisé dans les fonctions réflexes :

```
var results = jQuery("#results").html("<li>Loading...</li>");

jQuery.get("test.html", function(html){
    results.html(html);
});
```

Très utilisé dans les timers :

```
var count = 0;

var timer = setInterval(function(){
    if ( count < 5 ) {
        count++;
    } else {
        clearInterval( timer );
    }
}, 100);
```

## Propriété privée avec une fermeture

```
function T(){
  var x = 0;
  this.getX = function(){
    return x;
  };
  this.X = function(){
    x++;
  };
}
js> var t=new T()
js> t.x == undefined
true
js> t.getX()
0
js>t.X()
js>t.getX()
1
```

# Modules

```
(function() {  
  
    // declare private variables and/or functions  
    return {  
        // declare public variables and/or functions  
    }  
})();
```

```
var counter = (function(){
  var x = 0;
  function _inc(){
    x++;
  }
  function _dec(){
    x--;
  }
  return {
    INC:_inc,
    DEC:_dec,
    GET:function(){
      return x;
    },
    SET:function(a){
      x=a;
    }
  }
})();

counter.SET(0);
counter.INC();
counter.DEC();
console.log(counter.GET());
```

# Propriété prototype

On ajoute une propriété à tous les "instances" de la "classe" UN en passant par le prototype du "constructeur" (la fonction UN).

```
js> function UN(){this.un=1;}
js> var a=UN();
js> UN.prototype.deux=function(){return this.un+1}
js> var b=UN();
js> a.deux();
2
js> b.deux();
2
```

- Toute objet possède une propriété `__proto__` initialisé à la création avec le prototype du constructeur. (Celui-ci est vide initialement)
- Si un attribut n'est pas disponible au niveau de l'objet, javascript regarde dans l'objet `__proto__`, et ainsi de suite.



# Un autre exemple

```
function Rectangle(largeur, hauteur) {  
  this.largeur = largeur;  
  this.hauteur = hauteur;  
}  
  
Rectangle.prototype = {  
  surface: function() {  
    return this.largeur * this.hauteur;  
  },  
  
  perimetre: function() {  
    return (this.largeur + this.hauteur) * 2;  
  }  
};
```

# Héritage

Il ne s'agit pas vraiment d'héritage au sens de la POO, mais d'un chaînage par le prototype du constructeur.

Un exemple : On crée une "classe" Point :

```
function Point(x,y){
  this.x=x;
  this.y=y;
  this.s=function(){
    return (this.x+this.y)/2}
}
js>p=new Point(1,2);
js>p.s()
1.5
```

On veut rajouter à notre "classe" point un nom, pour la dériver en PointN. Comment faire ?

On pourrait "naïvement" utiliser dans le constructeur de la classe "dérivée" le constructeur de la classe parente :

```
function PointN(x,y,nom){
  Point.call(this,x,y);
  this.nom=nom;
}
js>q=new PointN(1,2,"A");
js>q.s()
1.5
```

Cela fonctionne .... presque !

```
js> Point.prototype.zero=function(){
  this.x=0;
  this.y=0;
}
js> q.zero()
typein:34: TypeError: q.zero is not a function
js>q instanceof Point
false
```

- Javascript ne trouve pas la propriété zero dans l'objet q.
- Il cherche dans l'objet `__proto__`, qui est le prototype de `PointN`, qui est vide !

Comment faire ?

Avant "d'instancier la classe PointN", on modifie le prototype du constructeur avec un objet Point.

```
PointN.prototype=new Point();
```

Ainsi, Lorsque on demande une propriété définie dans Point, Javascript regarde dans l'objet `__proto__` qui est un point. Le chaînage est assuré !

# Fonctions réflexes et paramètres

---

# Problème et solutions

Lorsque l'on enregistre une fonction réflexe en réponse à un événement, on a pas la maîtrise des paramètres qui leurs sont envoyés. En effet, c'est le gestionnaire d'événement qui donne les paramètres au moment de l'appel. Celui-ci se limite d'ailleurs à ne donner qu'un paramètre, une référence à un objet Event. Typiquement :

```
elm.onclick = function ReponseClick(event){  
  //traitement  
}
```

Comment transmettre à la fonction des données supplémentaires ?

# Avec les fermetures

La fonction réflexe peut accéder à une variable déclarée en dehors de celle-ci. Le problème se pose lorsque cette variable change. Par exemple :

```
for(i=0;i<10;i++){  
  document.images[i].onclick=function(){  
    alert("i vaut "+i);  
  }  
}
```

Que se passera-t'il ?



La première solution consiste à enregistrer la fonction réflexe comme le retour d'un appel à une fonction qui va capturer les données que l'on souhaite accéder au moment de l'exécution de la fonction réflexe.

```
for(i=0;i<10;i++){
  document.images[i].onclick=(function(x){
    return function (event){
      alert("i vaut "+x);
    }
  })(i);
}
```

- Nous sommes en présence d'une fermeture et d'une fonction anonyme imbriquée. Celle-ci a accès aux variables et paramètres de la fonction imbriquante.
- Ainsi il est possible d'attacher une fonction événementielle paramétrée. La fonction interne sera appelée au moment de l'évènement mais elle aura toujours accès aux variables et paramètres de sa fonction imbriquante quand bien même celle-ci a terminé son exécution depuis.

# Binding

On utilise la méthode `bind` de la "classe" `Function`.

```
for(i=0;i<10;i++){  
  document.images[i].onclick=(function(){  
    var i=this.i;  
    alert("i vaut "+i);  
  }).bind({"i":i});  
}
```

La fonction `bind()` crée une nouvelle fonction qui, lorsqu'elle est appelée, a pour contexte `this` la valeur passée en paramètre et éventuellement une suite d'arguments qui précéderont ceux fournis à l'appel de la fonction créée.

objet

{ }

{ membres }

membres

chaîne : valeur

membres , chaîne : valeur

tableau

[ ]

[ éléments ]

éléments

valeur

éléments , valeur

valeur

chaîne

nombre

objet

tableau

true

false

null