Javascript , ECMAScript, JScript, ... : kesako ?

- **ECMAScript (ES)**
  Nom du langage spécifié et normalisé (par ECMA).
  Évolution majeure : ES2015 (ES6)
  Version la plus récente : ES2016 (ES7)
- **JavaScript**
  - Nom historique du langage (Brendan Eich, 1995)
  - Nom de l'implémentation dans Mozilla, Chrome, Safari...
- **JScript**
  Nom de l'implémentation Microsoft
- **ActionScript**
  Nom de l'implémentation Adobe
- **Node.js**
  Javascript côté serveur Web

Javascript (ECMAScript) : un langage Objet

La plupart des valeurs natives sont des objets ou peuvent être manipulés comme tels

- tableaux (`Array`)
- fonctions (`Function`)
- `String` (wrapper de chaîne)
- `Number` (wrapper de nombre)
- `Boolean` (wrapper de booléen)

Les bibliothèques définissent de nombreux objets

- en particulier l'interface DOM (nœuds et éléments du document sont des objets)

Javascript : un langage Objet

Des objets mais pas de classes

- Le modèle objet de javascript se fonde sur les **prototypes** et non sur les **classes**
- Un objet possède un prototype : autre objet qui est son «modèle».
- Il n'existe pas de classe en JS même si parfois on utilise ce mot de façon impropre.

Nouveauté ES2015

Cette version ajoute la possibilité d'écrire des **pseudo-classes** et même celle d'utiliser du **pseudo-héritage**.
Progrès en terme d'écriture de code,mais il ne s'agit que de «sucre syntaxique» (nouveauté de forme, pas de fond).
Implémentée récemment par les navigateurs, des problèmes de compatibilité pourront encore être rencontrés par de nombreux utilisateurs

La pseudo-classe Object

Exemple simpliste

```
x = {}; // équivaut à
x = new Object();
```

Objet avec attributs :

```
x = { att1 : 12, att2 : "abcd"};
x.att1; // vaut 12
```

Objet avec méthode :

```
x = { att1 : 12, att2 : "abcd",
 concat  : function(){
    return this.att1 + this.att2;
 }
};
x.concat(); // renvoie "12abcd"
```


Ajout et suppression dynamique

- On peut à tout instant ajouter une propriété à un objet.

```
p.nomMaj = function(){return this.nom.toUpperCase()};
p.nomMaj(); // renvoie "DUPONT"
```

- Attention : l'ajout ne concerne **que** l'objet p, pas les autres instances de `Personne`
- Suppression d'une propriété possible par la commande `delete`

Une méthode est une propriété (attribut)

- Une méthode est une propriété dont la valeur est une fonction.
- Cette fonction s'exécute **« dans le contexte de l'objet »** : `this` désigne l'objet auquel elle appartient.

Javascript : un langage Objet

Les fonctions utilisées comme constructeur

- Constructeur : fonction d'initialisation.
- Array, Number, String sont en réalité des fonctions.
- `instanceof`

```
var t = [8,5,12];
t instanceof Array   // vaut true
```

Exemple

```
function MonObjet(n, s){
  this.att1 = n;
  this.att2 = s;
  this.concat  = function(){ // on verra mieux par la suite
    return this.att1 + this.att2;
 }
}
x = new MonObjet(4,"xyz");
x.concat(); // renvoie "4xyz"
```

Le prototype

Javascript : modèle d'héritage par prototype

- chaque objet possède un prototype, qui est un autre objet (ou éventuellement null)
- quand on cherche à consulter la propriété (attribut) d'un objet, si celle-ci est absente, JS cherche la propriété de même nom de son prototype puis du prototype du prototype ... etc
- ⇒ c'est la **chaîne de prototypes**

```
//on suppose  :
// obj1 vaut {a:1, b:2, c:3}, obj2 vaut {b:20,d:40};
// et obj1 est prototype de obj2 (on verra comment faire er
obj2.d; // 40
obj2.c; // 3
obj2.b; // 20
```


Exemple de pseudo classe (syntaxe ES5)

```
1  /* Constructeur */
2  function Ratio(num, den){
3     if (den===0)
4        throw "dénominateur nul";
5     this.num = den >0 ? num : -num;
6     this.den = den >0 ? den : -den;
7     this.reduce();
8  }
9  /* Méthodes */
10 Ratio.prototype.toFloat = function(){
11    return this.num/this.den;
12 };
13 Ratio.prototype.reduce = function(){
14    var div = Ratio.pgcd(this.num, this.den);
15    if (div>1){
16       this.num /= div;
17       this.den /= div;
18    }
19 };
```


Exemple de pseudo classe (syntaxe ES5)

```
20 /* Méthode "static" */
21 Ratio.pgcd = function(a,b){
22    while (a != b){
23       if (a>b)
24          a = a-b;
25       else
26          b = b-a;
27    }
28    return a;
29 };
```

```
1  x = new Ratio(6,4);
2  x.num; // vaut 3
3  x.den; // vaut 2
4  x.toFloat(); // renvoie 1.5
```


Exemple de pseudo classe (syntaxe ES2015)

```
1  class Ratio6{
2     /* Constructeur */
3    constructor (num, den){
4        if (den===0)
5           throw "dénominateur nul";
6        this.num = den >0 ? num : -num;
7        this.den = den >0 ? den : -den;
8        this.reduce();
9     }
10    /* Méthodes */
11    toFloat (){
12       return this.num/this.den;
13    }
14    reduce (){
15       var div = Ratio.pgcd(this.num, this.den);
16       if (div>1){
17          this.num /= div;
18          this.den /= div;
19       }
```


Exemple de pseudo classe (syntaxe ES2015)

```
20    }
21    /* Méthode "static" */
22    static pgcd (a,b){
23       while (a != b){
24          if (a>b)
25             a = a-b;
26          else
27             b = b-a;
28       }
29       return a;
30    }
31 }
```

```
1  x = new Ratio6(6,4);
2  x.num; // vaut 3
3  x.den; // vaut 2
4  x.toFloat(); // renvoie 1.5
```


Mise à jour dynamique des pages

(Rappel) Un outil : les timer

Les timer permettent un déclenchement différé ou répétitif d'une fonction.

- `var timeoutId = setTimeout(maFonction,delai)` : va déclencher maFonction au bout de `delai` ms
- `clearTimeout(timeoutId)` : annule le déclenchement différé.
- `var intervalId = setInterval(maFonction,delai)` : va déclencher maFonction toutes les `delai` ms
- `clearInterval(intervalId)` : annule le déclenchement périodique.

Mise à jour dynamique des pages

Lancer des requêtes HTTP : l'objet XMLHttpRequest

- Le script JS pourra envoyer une requête HTTP
- puis récupérer le résultat de cette requête
- de façon invisible pour l'utilisateur
- les données reçues peuvent être exploitées pour mettre à jour une partie de la page courante, via le DOM

Synchrone ≠ Asynchrone

Deux modes de fonctionnement :

- mode **synchrone** : le script s'arrête et attend la réponse. Pendant ce temps l'interface utilisateur est perturbée ⇒ **INADAPTÉ** dans la quasi-totalité des cas.
- mode **asynchrone** : le script continue sans attendre la réponse. L'arrivée de la réponse est un **évènement** → à traiter par un gestionnaire d'évènement.

Mise à jour dynamique des pages

Étapes

- Instancier un objet XMLHttpRequest
- Préparer la requête
- Mettre en place les gestionnaires d'évènement
- Envoyer la requête

```
var requ = new XMLHttpRequest();
requ.open("GET","http://www.monsite.fr/x/y",true);
requ.addEventListener("load",traiteReponse);
requ.addEventListener("error",traiteErreur);
requete.send(null);
```


Mise à jour dynamique des pages

Traitement de la réponse

Les données reçues sont disponibles dans un attribut de l'objet XMLHttpRequest

- `responseText` **: données sous forme de texte**
- `responseXML` : objet XMLDocument (si données XML)
- `response` : autres types d'objets, notamment binaires (positionner d'abord responseType)
  (pas encore totalement implémenté par les navigateurs)

Mise à jour dynamique des pages

Exemple

On suppose que la réponse est un texte simple.
On veut le placer dans un paragraphe `p#message` préexistant dans le document.

```
function traiteReponse(ev){
  var parag = document.querySelector("p#message");
  parag.textContent = this.responseText;
}
```

this

`traiteReponse` est un gestionnaire d'évènement associé à l'objet `XMLHttpRequest`
Il s'exécute donc **dans le contexte de cet objet**
⇒ `this` désigne l'objet `XMLHttpRequest` qui a lancé la requête

Mise à jour dynamique des pages

Exemple

On suppose que la réponse est un tableau de chaînes, au format JSON.
Ces chaînes sont à ajouter dans une liste `ul#liste` préexistante dans le document.

```
function traiteReponse(ev){
  var tab = JSON.parse( this.responseText );
  var liste = document.querySelector("ul#liste");
  for (var i=0; i<tab.length; i++){
    var item = document.createEelement("li");
    item.textContent = tab[i];
    liste.appendChild(item);
  }
}
```


Mise à jour dynamique des pages

Autres **attributs** de XMLHttpRequest

- `status` : code réponse HTTP (chaîne, ro)
- `statusText` : texte réponse HTTP (chaîne, ro)
- `readyState` : code d'état *(plus très utile)* (entier, ro)
- `timeout` : fixe un timeout (millisecondes) à la requête (nombre, rw)
- `withCredentials` : autorise la présentation des cookies sur un site tiers (booléen, rw)

Autres **méthodes** de XMLHttpRequest

- `setRequestHeader(nom de propriété, valeur)` : fixe une propriété dans l'en-tête HTTP de la **requête**
- `getResponseHeader(nom de propriété)` : valeur de la propriété dans la **réponse** HTTP
- `getAllResponseHeaders()` : ensemble des entêtes HTTP

Mise à jour dynamique des pages

Évènements de XMLHttpRequest

- `load` : la réponse est complète
- `error` : échec de la requête
- `timeout` : abandon suite à un timeout
- `loadstart` : la réception de la réponse commence
- `progress` : progression dans la réception des données

Mise à jour dynamique des pages

Sites tiers (cross-domain)

- Initialement (XMLHttpRequest 1) le concept est prévu pour n'envoyer des requêtes que vers le site d'origine (celui d'où vient le document web courant)
- Le cross-domain est maintenant possible sous conditions : en particulier le site destinataire doit autoriser explicitement l'opération.
  Le site destinataire doit positionner la propriété `Access-Control-Allow-Origin`
- Plusieurs limitations dues à la confidentialité et la sécurité

Mise à jour dynamique des pages

Quelles format de données échanger ?

- XML (non traité dans ce cours)
- texte simple ou semi structuré (ex : un nombre, une date, une ligne CSV, ...) uniquement pour des données très courtes ou simples.
- (X)HTML ou un fragment de (X)HTML : portion de document
  - faciles à intégrer dans le document (via innerHTML par exemple)
  - peu évolutif : la structure HTML envoyée doit être intégrée telle quelle par le client
- JSON données structurées simples
  - faciles à convertir en valeur JS
  - données assez compactes
  - demande un peu de code JS pour générer les objets DOM.

Web service

- Application accessible en réseau via le protocole Web (HTTP)
- Fournit des données à une autre application
  - sur un autre point réseau
  - contexte en général hétérogène (langage, système, environnement ...)
- Ce que produit l'application n'est pas, *a priori*, destiné à être consulté directement par l'utilisateur final, ni même par un «humain»
- Les productions de type JSON ne sont que le niveau le plus simple des Web Services
- Il existe des protocoles de haut niveau pour structurer les échanges : REST, SOAP, WSDL
  $\Rightarrow$ vers la notion d'**application distribuée**