Développement Web

Javascript

LI328
Laure Soulier
slides de Sylvain Lamprier

UPMC

Scripts Web

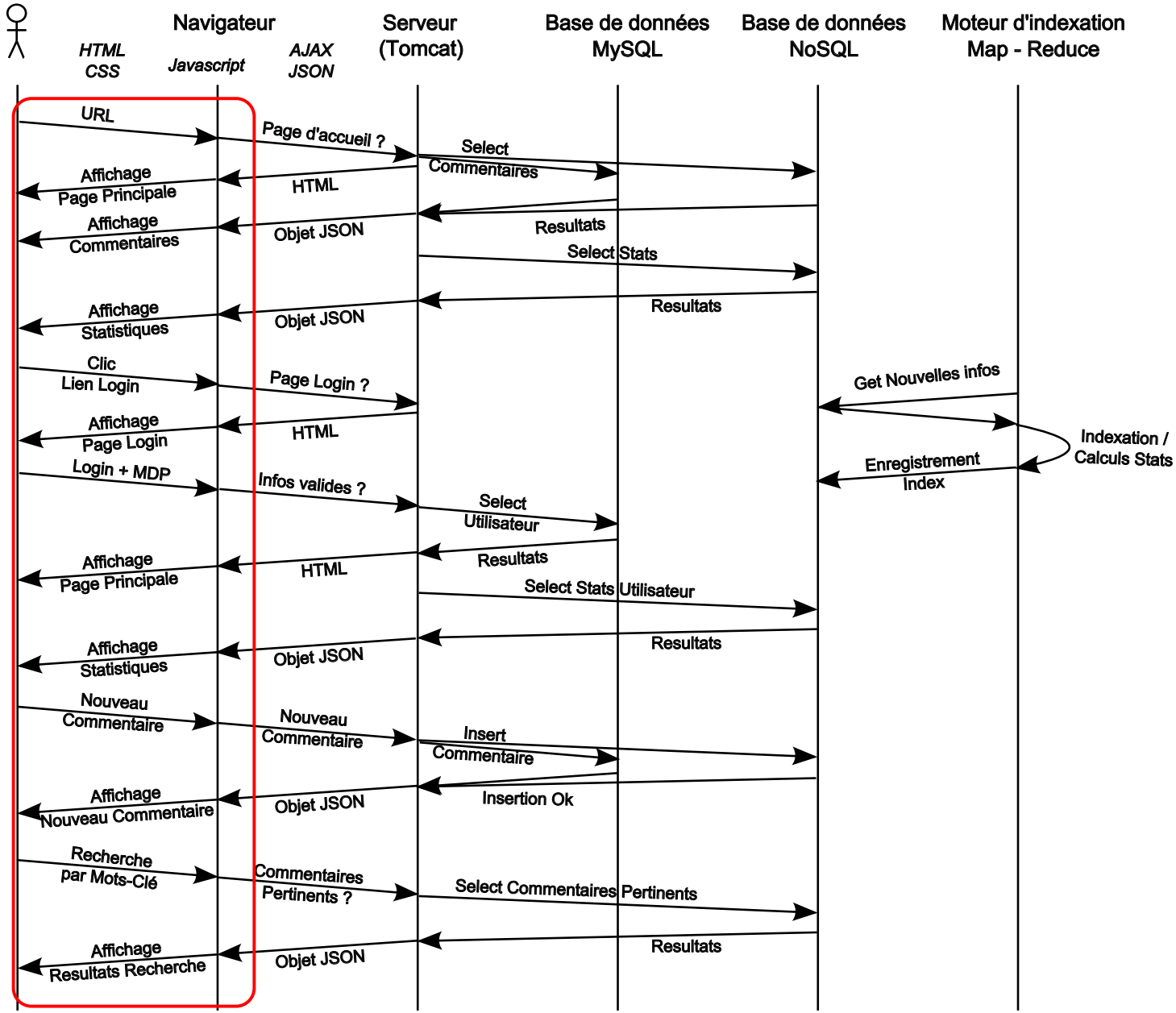
- HTML / CSS permettent de produire des pages Web avec dispositions graphiques évoluées mais
  - Pages peu dynamiques
  - Interactions utilisateur limitées
- Langages de script pour le Web
  - Permettent de combler ce manque en donnant la possibilité de définir des fonctions
    - De modification des éléments affichés (modification éléments / attributs / propriétés de l'arbre DOM)
    - De réaction à des actions utilisateur
    - De communication client / serveur
  - Exécution de traitements côté client
  - Fortement dépendants du navigateur appelant la page web

Scripts Web

- Différents langages
  - Javascript (Netscape & Sun)
  - VBscript (Microsoft)
  - XUL (Mozilla)
  - XSLT (W3C)

- European Computer Manufactures Association (ECMA)
  - ⇒ Standard ECMA 262

Scripts Web


Navigateur
HTML
CSS
Javascript
AJAX
JSON
Serveur
(Tomcat)
Base de données
MySQL
Base de données
NoSQL
Moteur d'indexation
Map - Reduce
URL
Page d'accueil ?
Select
Commentaires
Affichage
Page Principale
HTML
Affichage
Commentaires
Objet JSON
Resultats
Select Stats
Affichage
Statistiques
Objet JSON
Resultats
Clic
Lien Login
Page Login ?
Get Nouvelles infos
Affichage
Page Login
HTML
Indexation /
Calculs Stats
Login + MDP
Infos valides ?
Enregistrement
Index
Select
Utilisateur
Affichage
Page Principale
HTML
Resultats
Select Stats Utilisateur
Affichage
Statistiques
Objet JSON
Resultats
Nouveau
Commentaire
Nouveau
Commentaire
Insert
Commentaire
Affichage
Nouveau Commentaire
Objet JSON
Insertion Ok
Recherche
par Mots-Clé
Commentaires
Pertinents ?
Select Commentaires Pertinents
Affichage
Resultats Recherche
Objet JSON
Resultats

- Javascript ≠ Java
  - Java : Langage objet compilé, qui peut être exécuté côté client par le biais d'une applet
  - Javascript : Langage de scripts moins évolué, intégré aux pages Web, qui s'inspire de différents langages (dont Java)

Javascript	**Applet Java**
Langage interprété	Langage compilé
Interprétation par le navigateur	Chargement d'une machine virtuelle
Code intégré au HTML	Code appelé à partir de la page
Langage peu typé	Langage fortement typé
Accessibilité du code	Confidentialité du code

- Javascript
  - Langage orienté objet à héritage prototypique
    - Objets pas instances de classes
    - Prototypes d'objets
    - ⇒ En javascript, toute entité, fonction ou structure peut être vue comme un objet

  - ⇒ Définition de fonctions se déclenchant
    - Au chargement de la page
    - En réaction à des évènements de la page

Évènements

Gestionnaire d'évènement	Description
onClick	Lors d'un clic sur l'élément associé à l'événement
onLoad	Lorsque le navigateur charge la page en cours
onUnload	Lorsque le navigateur quitte la page en cours
onMouseOver	Lorsque le curseur de la souris passe au-dessus de l'élément
onMouseOut	Lorsque le curseur de la souris quitte l'élément
onFocus	Lors de l'obtention du focus (élément sélectionné comme étant l'élément actif)
onBlur	Lors de la perte du focus (l'utilisateur clique hors de l'élément)
onChange	Lors de la modification du contenu d'un champ de données
onSelect	Lors de la sélection d'un texte (ou une partie d'un texte) dans un champ de type "text" ou "textarea"
onSubmit	Lors d'un clic sur un bouton de soumission d'un formulaire (le bouton qui permet d'envoyer le formulaire)

Insérer du code Javascript

Javascript dans balises

```
<!-- Pour différentes balises (avec eventHandler le nom
d'un gestionnaire d'événement) -->
<balise eventHandler="javascript:(function(){...})()">

<!-- ou en tant qu'action d'un formulaire -->
<form action="javascript:(function(){...})()">
```

Javascript dans <script> ... </script>

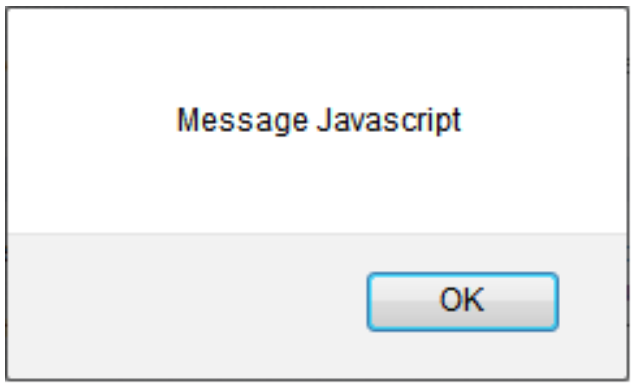
```
<script type="text/javascript"> Code javascript </script>
```

Javascript dans un ou plusieurs fichiers séparés

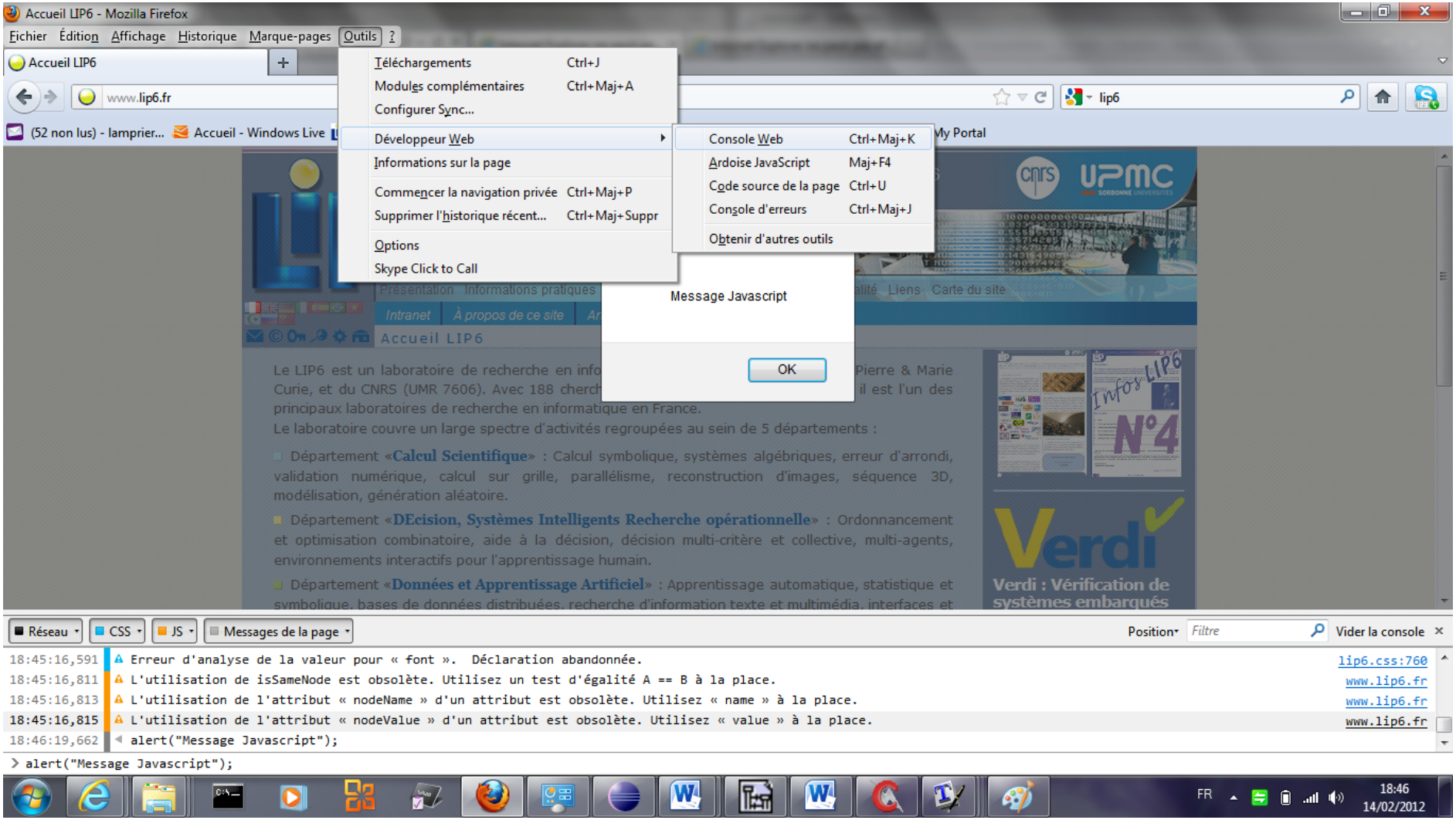
```
<script type="text/javascript" src="scripts/code.js"></script>
```

Exemple premier code Javascript

```
<HTML>
<HEAD>
<TITLE>Exemple Javascript</TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
<!--
alert("Message Javascript");
// -->
</SCRIPT>
</BODY>
</HTML>
```



Debuguer du Javascript : Console Web



Javascript

1. Le langage Javascript
2. Le format JSON
3. Manipulation d'objets DOM
4. La librairie JQuery

Javascript : Les variables

- Variables :
  - Conteneurs pouvant accueillir toutes sortes de choses
  - En Javascript, pas de type défini lors de la déclaration
- Deux types de variables
  - Variables à portée locale
    - Variable déclarée à l'intérieur d'une fonction et avec le mot-clé *var*

    ⇒ Visible uniquement dans la fonction où elle est déclarée
  - Variables à portée globale
    - Variable déclarée à l'extérieur d'une fonction ou sans le mot-clé *var*

    ⇒ Visible partout

Javascript : Les variables

Variables locales / Variables globales

```
i = 0; // Variable globale
var j = 'Technos Web'; // Variable globale

function f(){
  var k = 'Javascript'; // Variable locale
  l = 20; // Varibale globale (car sans le mot-clé var)
  return i*l; // x est accessible ici car il s'agit d'une variable globale
}
// k n'est plus accessible ici (car locale)
```

Javascript : Les Fonctions

- Une fonction est un bloc d'instructions acceptant une liste de paramètres
  - Définie par le mot-clé *function*
  - Peut posséder un nom
  - Retourne éventuellement une valeur (grâce à *return*)

Fonction Javascript

```
function nom_fonction(parametre_1, ... , parametre_n) {
    instructions;
    return expression;
}
```

- On peut stocker une fonction dans une variable

Fonction anonyme

```
var f = function(parametre_1, ... , parametre_n) {
    instructions;
    return expression;
};
```

Javascript : Les Fonctions

- Paramètres de fonctions
  - Pas obligatoirement même nombre de paramètres à l'appel et dans la définition
    - Affectation des paramètres dans l'ordre
    - Les paramètres peuvent être accédés par l'objet arguments ⇒ *arguments[n-1]* correspond à la n-ième valeur passée en paramètre
    - Si nombre de paramètres à l'appel < nombre de paramètres attendus ⇒ paramètres supplémentaires = *undefined*
    - Si nombre de paramètres à l'appel > nombre de paramètres attendus ⇒ paramètres additionnels peuvent être récupérés par l'objet arguments
  - Les paramètres sont passés par référence sauf les types de bases (nombres, caractères,...) qui sont passés par valeur (copie)

Paramètres

```
function fonc(x) {
    alert("x = "+x+" y= "+arguments[0]+", z ="+arguments[1]+", w = "+arguments[2]);
}
fonc(1,2); // Affiche x=1, y=1, z=2, w=undefined
fonc(1); // Affiche x=1, y=1, z=undefined, w=undefined
fonc(); // Affiche x=undefined, y=undefined, z=undefined, w=undefined
```

- Une fonction peut être définie à l'intérieur d'une autre
  - La fonction interne peut accéder aux variables locales de la fonction externe...
  - ... Et s'en souvient même après en être sortie

  ⇒ On parle de fermeture

Fermeture

```
var x = 1;
function fonc() {
    var x=2;
    y = function() {alert(x);}
    alert(x);
}
alert(x);       //  affiche "1" (car le x de fonc est local)
fonc();         // affiche "2" (normal, on est dans fonc)
y();            // affiche "2" et pas "1" (bien que l'on soit hors de fonc)
```

Javascript : Les Tableaux

- Un tableau est un ensemble d'éléments repérés par leur indice (entier commençant à 0)
  - Création
    - Création littérale
      Exemple : *var tab = [0,1,,,4,5];*
    - Création grâce au constructeur Array
      Exemple : *var tab=new Array(10);*
  - Accès aux éléments par la notation tableau[indice]
    Exemple : *tab[0]=10;* *var x=tab[0];*
  - Propriété length qui représente la longueur du tableau
    - L'ajout d'un élément à un indice supérieur à cette taille, augmente automatiquement length
    - Décrémenter length revient à supprimer le dernier élément
  - Méthodes spécifiques comme join, slice, et push

Javascript : La notion d'objets

- Contrairement aux langages objets classiques (Java, C++, ...)
  - La notion de classe n'existe pas
  - Le langage n'est pas typé
- En JavaScript, un objet correspond à une sorte de tableau associatif
  - Chaque élément d'un objet correspond à une "entrée" (identifiée par un nom)
    - Un attribut correspond à une entrée avec un type quelconque
    - Une méthode correspond à une entrée dont le type attendu est fonction
  - Manipulation dynamique
    - Possibilité d'ajouter, modifier ou supprimer les entrées de l'objet tout au long de sa vie
  - Mais pas de *length*, ni fonctions *join*, *splice*, *push*, etc...

Javascript : Les objets

- Création d'objets
  - Deux modes de création
    - Création à l'aide de constructeur
    - Construction littérale
- Création par constructeur
  - Un constructeur est une fonction qui associe des valeurs (pouvant être des fonctions) à des attributs
  - Un constructeur est appelé par le mot-clé *new*
  - Constructeur prédéfini *Object*
    Exemple : *var obj=new Object();*
- Construction littérale
  - Déclaration entre {...} d'associations attributs-valeurs
    Exemple : *var obj={x : 1, y : 'chaine'};*
  - Les valeurs associées peuvent elles-mêmes être des objets
    - ⇒ Base du format JSON

Javascript : Les objets

- Les propriétés des objets peuvent être créées, lues et écrites
  - Soit avec la notation "point" objet.propriété :
    Exemple : *var x=obj.couleur;*
  - Soit avec la syntaxe utilisée pour les éléments de tableau :
    Exemple : *var x=obj['couleur'];*

Objets Javascript

```
var obj = new Object();

obj["attribut"] = "valeur1";
// ou
obj.attribut = "valeur1";

var fonc=function(parametre1, parametre2) {
    alert("parametres: " + parametre1 + ", " + parametre2);
};

obj["methode"]=fonc;
// ou
obj.methode=fonc;

// Affichage de la valeur "attribut" de obj
alert("Valeur de attribut: " + obj.attribut);

// Exécution de la méthode methode de obj
obj.methode("valeur1", "valeur2");
```

- Constructeurs
  - Simples fonctions d'association propriété-valeur
  - Pas de notions de classes comme dans autres langages
  - Par convention, noms de constructeurs commencent par une majuscule

- Possibilité d'écrire ses propres constructeurs d'objets
  - Utilisation du mot-clé this
    - this se rapporte à l'objet sur lequel a été appelé la fonction
      Exemple : *l'objet x dans x.a()*
    - Dans le cas d'un constructeur, appel avec new

    ⇒ this se rapporte alors à l'objet nouvellement créé
  - Affectation de valeurs à différentes propriétés de l'objet
    Exemple : *this.couleur=2;*

Constructeur d'objets

```
function MonObjet(param1, param2) {
  this.attribut1 = param1;
  this.attribut2 = param2;
}

objet = new MonObjet(5, 'bleu');

alert(objet.attribut1);       // affiche 5
alert(objet["attribut2"]);    // affiche "bleu"

objet.attribut3 = new Date();  // ajoute une nouvelle propriété à l'objet
alert(objet.attribut3);       // affiche la date d'ajout de attribut3

MonObjet.statique = "serif";   // ajoute une propriété statique
alert(MonObjet.statique);     // affiche "serif"
alert(objet.statique);       // affiche undefined

delete objet.attribut2;         // enlève une propriété à l'objet
alert(objet.attribut2);       // affiche undefined

delete objet;                   // supprime l'objet entier (rarement utilisé)
alert(objet.attribut1);       // déclenche une exception
```

Javascript : Les objets

- *this* permet d'affecter des attributs / méthodes à un objet
  - On peut y accéder de l'extérieur du constructeur
  - ⇒ Ils sont donc publiques
- Grâce au mécanisme de fermeture, on peut définir des attributs / méthodes privés :

Constructeur d'objets

```
    function MonObjet(val) {
    var a = val;
    this.getA = function() {
      return(a);
    }
    this.setA = function(newVal) {
      a = newVal;
    }
}
var obj = new MonObjet(1);
console.log("A: " + obj.a);  // Affiche undefined (a non défini)

console.log("A: " + obj.getA()); // Affiche "A: 1" (getA est un accesseur)

obj.a=2;
console.log("A: " + obj.getA()); // Affiche "A: 1" (la valeur n'a pas changé)

obj.setA(2);
console.log("A: " + obj.getA()); // Affiche "A: 2" (setA est un modificateur)
```

- On peut faire la même chose avec les méthodes :

Constructeur d'objets

```
function MonObjet(parametre1, parametre2) {
    var attribut1 = parametre1;
    var attribut2 = parametre2;
    var methode_privee = function() {
        console.log("Attributs: " + attribut1 + ", " + attribut2);
    }
    this.methode_publique = function() {
        methode_privee();
    }
}
var obj = new MonObjet(1, 2);
alert("Attribut1: " + obj.attribut1);  // Affiche undefined (attribut1 non défini)

obj.methode_privee(); // Affiche "TypeError: obj.methode_privee is not a function"
obj.methode_publique(); // Affiche "Attributs: 1, 2"
```

Javascript : Prototypage

- Ce processus de construction d'objet est relativement simple mais...
  - A chaque fois nouvelle création avec un constructeur contenant n fonctions internes, n fonctions sont créées pour le nouvel objet
    - ⇒ Créer 10 objets avec un constructeur contenant 10 fonctions revient à la création de 100 fonctions
    - ⇒ Peut vite devenir coûteux (d'autant plus avec des objets composites)
    - ⇒ Le comportement souhaité serait que tous les objets pointent vers la méthode methode.
  - ⇒ Le mécanisme de prototypage de permet de pallier à cet problème

Javascript : Prototypage

- Propriété *prototype* du constructeur
  - Accueille un objet définissant les données par défaut
  - ⇒ Affectations faites pour l'objet (avant ou après déclaration prototype) ont priorité sur les affectations du prototype
  - ⇒ Les attributs d'un objet sont accessibles dans un prototype par *this.* uniquement pour attributs publiques (privé impossible dans ce cas)
  - ⇒ Tous les objets créés par un constructeur utilisant un prototype pointent vers les mêmes fonctions qui y sont déclarées

Prototypes d'objets

```
function MonObjet(parametre1, parametre2) {
    this.attribut1 = parametre1;
    this.attribut2 = parametre2;
}
MonObjet.prototype.attribut1=1;
MonObjet.prototype.attribut3=3;
MonObjet.prototype.methode = function() {
  alert("Attributs: " + this.attribut1 + ", " + this.attribut2+", "+this.attribut3);
}
var obj = new MonObjet(5, 4);
MonObjet.prototype.attribut2=2;

obj.methode();
// Affiche "Attributs: 5, 4, 3"
//(les valeurs prototypiques pour attributs 1 et 2 n'ont aucun effet
// car priorité aux valeurs données au constructeur)
```

Javascript : Prototypage

- Possibilité d'inclure la création du prototype dans le constructeur
  - ⇒ Permet d'avoir accès à toutes les variables du constructeur
  - ⇒ Mais attention à ne le définir qu'une seule fois !
  - ⇒ Mais attention valeurs d'attributs privés définies à la première création !

Prototypes d'objets

```
function MonObjet(parametre1, parametre2) {
    this.attribut1 = parametre1;
    var attribut2 = parametre2;
    if( typeof MonObjet.initialized == "undefined" ) {
        MonObjet.prototype.methode = function() {
            alert("Attribut: " + this.attribut1 + attribut2);
        };
        MonObjet.initialized = true;
    }
}
var obj = new MonObjet(1, 2);
obj.methode(); // Affiche "Attributs: 1, 2"
var obj2 = new MonObjet(3, 4);
obj2.methode(); // Affiche "Attributs: 3, 2"
```

Javascript : Héritage

- Le prototypage permet de mettre en place un forme d'héritage propre aux langages objets
  - ⇒ Prototype d'un constructeur = Objet construit avec un autre constructeur

Héritage par prototype

```
function m1() {return 1;}
function m2() {return 2;}
function m3() {return 3;}
function Sub() {}
Sub.prototype.y = m2;

alpha = new Sub();
console.log(alpha.y());              // imprime 2
function Sur(){this.y = m3;}
parent = new Sur();
beta = new Sub();
Sub.prototype = parent;               // n'a pas d'effet sur alpha et beta  déjà créés
console.log(beta.y());                // imprime 2
gamma = new Sub();
console.log(gamma.y());              // imprime 3
parent.y = m1;                       // impacte gamma
console.log(gamma.y());              // Affiche 1

// A noter :
// Les changements de prototype n'ont pas d'effet sur les objets déjà créés
// Mais les changements dans le prototype en ont !
```

Javascript : Construction littérale

- Comme on l'a vu précédemment :
  - Un objet peut être instancié par un constructeur...
  - ... ou formé par une déclaration littérale
- Construction littérale :
  - Un objet se construit par { }
  - Un tableau se construit par []
  - Possibilité de former des imbrications d'objets / tableaux

Exemple de construction littérale

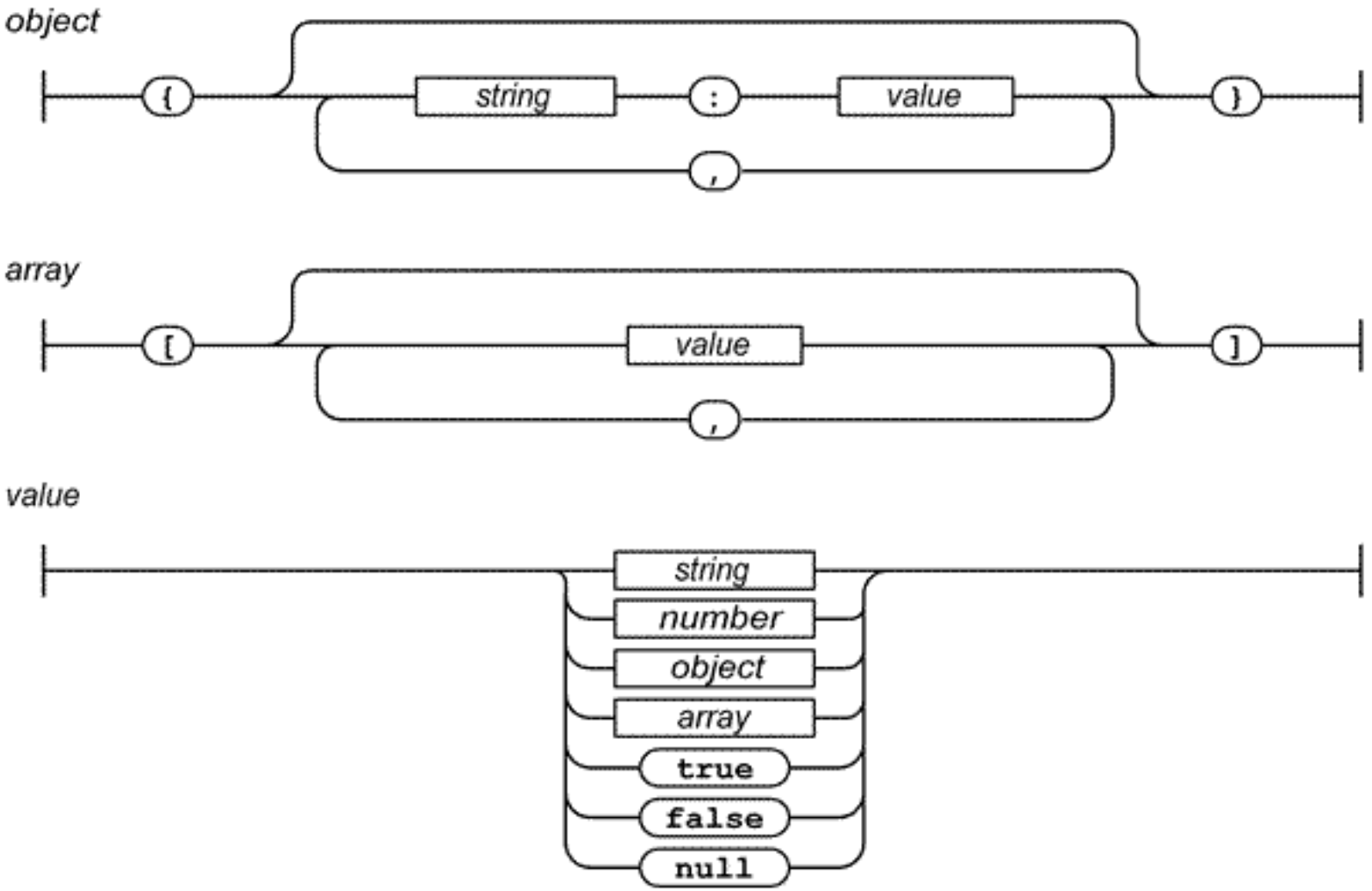
```
var obj={"menu": {
   "id": "file",
   "value": "File",
   "popup": {
     "menuitem": [
       {"value": "New", "onclick": function(){alert("Create New Doc");}},
       {"value": "Open", "onclick": function(){alert("Open Doc");}},
       {"value": "Close", "onclick":  function(){alert("Close Doc");}}
     ]
   }
 }}

obj.menu.popup.menuitem[1].onclick(); // Affiche "Open Doc"
```

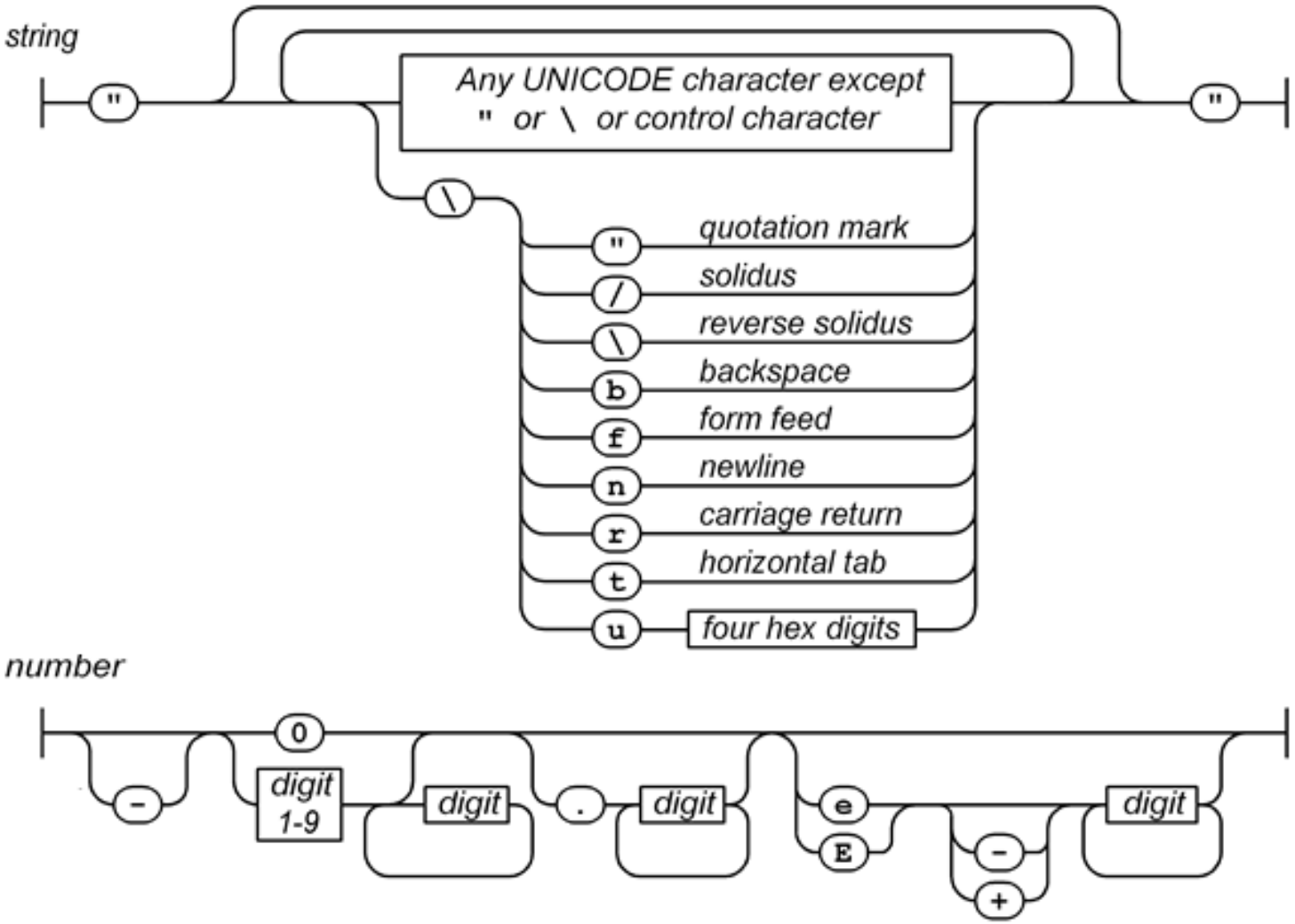
Javascript : Le format JSON

- JSON (JavaScript Object Notation)
  - Format de données textuel
  - Dérivée de la construction littérale d'objets
- Format JSON est composé :
  - d'ensembles de paires nom / valeur ⇒ les objets
  - de listes ordonnées de valeurs ⇒ les tableaux

Javascript : Le format JSON



Javascript : Le format JSON



Javascript : Le format JSON

- Le format JSON = chaine de caractères correspondant à la formation littérale d'un objet
  - ⇒ Nécessite de disposer :
    - D'un parser : texte JSON ⇒ objet
    - D'un serializer : objet ⇒ texte JSON

Javascript : Parser du JSON

- Parser : la fonction eval(string)
  - Permet d'interpréter une chaîne de caractères
  - Puisque le format JSON = chaine de construction littérale, *eval*('('+*json_text*+')') construit l'objet correspondant au texte contenu dans *json_text*
- Mais :
  - eval est une fonction générique permettant d'évaluer n'importe quelle chaîne représentant du code Javascript

  ⇒ Problèmes de sécurité car du code nuisible peut être exécuté lors de la transformation du texte JSON

Faille de sécurité

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\":\"json\"}";

var obj=eval("("+json_texte+")");  // construction de l'objet

// JSON tronqué lors du transfert :
json_texte="function(){alert('Hack!')})(";

var obj=eval("("+json_texte+")");  // Affiche "Hack!"
```

Javascript : Parser du JSON

- Depuis 2009, les navigateurs intègrent un support JSON comportant un fonction *parse(json_text, reviver)*
  - json_text : chaîne JSON à transformer
  - reviver (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la construction de l'objet

  ⇒ Méthode spécifique n'interprétant pas d'autre code qu'une chaîne de construction JSON

JSON.parse

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\":\"json\"}";

// construction de l'objet
var obj=JSON.parse(json_texte);

Pour un parsing fonctionnant sur n'importe quel navigateur :
var obj = typeof JSON !='undefined' ?
            JSON.parse(json_texte) : eval('('+json_texte+')');

// JSON tronqué lors du transfert :
json_texte="function(){alert('Hack!')})(";

// Affiche "SyntaxError: JSON.parse: unexpected keyword"
var obj=JSON.parse(json_texte);
```

Javascript : Serializer en JSON

- Le support JSON comporte égalemment une fonction *stringify(objet, replacer);*
  - objet : objet à transformer en chaîne JSON
  - replacer (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la structure de l'objet pour spécifier un traitement spécial

JSON.stringify

```
//Objet à serializer en JSON
var obj=new Object();
obj.g="1";
obj.f="json";

var json_text=JSON.stringify(obj);
// json_text contient "{\"g\":\"1\",\"f\":\"json\"}"
```

⇒ Attention : JSON.stringify ne serialise pas ce qui est dans le prototype

Exemples utilisation replacer et revival : objet Date()

```
var obj={g:1, r:new Date()};
obj; // Affiche "({g:1, r:(new Date(1329482734849))})" sur la console Web

json_text = JSON.stringify(obj, function (key, value) {
                return this[key] instanceof Date ? 'Date(' + this[key] + ')' : value;
});

obj2=JSON.parse(json_text, function (key, value) {
                var d;
                if (typeof value === 'string' &&
                   value.slice(0, 5) === 'Date(' &&
                   value.slice(-1) === ')') {
                        d = new Date(value.slice(5, -1));
                        if (d) {
                                return d;
                        }
                }
                return value;
});

obj2; Affiche "({g:1, r:(new Date(1329482734000))})" sur la console Web
```

Javascript : JSON

Pour s'assurer que JSON est bien pris en charge par le navigateur :

```
<!-- Au cas ou pas JSON sur browser -->
<script type="text/javascript" src=
    "https://github.com/douglascrockford/JSON-js/blob/master/json2.js">
</script>
```