PW6

Programmation Web

Enrica Duchi, Sylvain Perifel, Cristina Sirangelo

L3 Info - Université Paris Diderot

Javascript

quelques références et tutoriels utiles

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

http://www.w3schools.com/js/

Javascript

- Javascript : un langage de script pour dynamiser le contenu des pages web
- Un document HTML peut contenir des morceaux de code javascript

1. Ils peuvent se trouver à la fois dans le <head> et dans le <body> délimités par <script> </script> (éléments HTML non-visibles)

```
<!DOCTYPE html>
<html>
<head>
    ...
    <script>
      du javascript
    </script>
    ...

</head>
<body>
    ...
    <script>
      du javascript
    </script>
    ...

</body>
</html>
```

2. Ils peuvent également être insérés en tant que valeurs de certains attributs des éléments HTML

```
<!DOCTYPE html>
<html>
...

  <div onclick=" du javascript " >
  ...
  </div>

...


</html>
```

Javascript

- Alternative pour le cas 1 : code javascript chargé depuis un fichier externe

```
<!DOCTYPE html>
<html>
<head>
    ...
    <script src="myScript1.js"></script>
    ...

</head>
<body>
    ...
    <script src="myScript2.js"></script>
    ...

</body>
</html>
```

- Un script externe se comporte exactement comme si il était écrit à l'intérieur de l'élément <script> correspondant

HTML et Javascript

- Le code javascript contenu / chargé dans un document HTML est exécuté par le navigateur
  - les fragments contenus dans les éléments HTML `<script> </script>` sont exécutés dans l'ordre, pendant le chargement de la page

```
<!DOCTYPE html>
<html>
<head>
     ...
     <script>
       du javascript
     </script>
     ...

</head>
<body>
     ...
     <script>
       du javascript
     </script>
     ...

</body>
</html>
```

HTML et Javascript

- Le code javascript contenu / chargé dans un document HTML est exécuté par le navigateur
  - le javascript associé aux attributs est exécuté en réponse à un événement déclenchant

```
<!DOCTYPE html>
<html>
...

  <div onclick=" du javascript " >
  ...
  </div>

...


</html>
```

Exemple.
le *javascript* est exécuté par le navigateur en réponse à un click sur la région de la page occupée par la `<div>`

HTML et Javascript

- Dans tous les cas, les but de l'exécution du javascript est de changer le HTML affiché par le navigateur
- Cela permet de :
  - dynamiser le contenu / style des pages Web
  - créer des effets / animations
  - valider l'input de l'utilisateur
  - afficher des données
  - etc.

complètement coté client (sans ultérieurs requêtes au serveur)

Javascript et le DOM HTML

- Javascript supporte les objets
  - ce qui se révèle un instrument fondamental pour manipuler l'HTML
- Le Javascript exécuté sur un document HTML peut manipuler une structure de données qui représente le document HTML associé
- L'interface qui permet d'accéder à cette structure de données est appelé DOM(*Document Object Model*),

en gros :

- chaque élément du document HTML (i.e. un) est représenté comme un objet Javascript
- l'objet qui représente un élément HTML contient les objets représentants ses sous-éléments
- l'objet `document` est celui qui représente le document HTML entier
- les objets du DOM ont des propriétés et des méthodes qui permettent de les manipuler, et donc manipuler la page HTML

Javascript et le DOM HTML

- Exemple 1 (Javascript pur)
  - méthode `getElementById( id )` :
    renvoie l'objet représentant le sous-élément identifié par *id*

```
<!DOCTYPE html>
<html>
  <body>
  <div id="toto" >
    <p> Contenu original </p>
  </div>

  <script>
  var elem = document. getElementById("toto");
  </script>
  </body>
</html>
```

Après le chargement de la page, la variable Javascript `elem` a pour valeur l'objet représentant la `<div>` avec `id="toto"`

Javascript et le DOM HTML

- Exemple 2 : (Javascript pur)

```
<!DOCTYPE html>
<html> <body>

  <img id="img1"  src="mon_image.jpg" >

  <button onclick = "alert(document. getElementById('img1').src)" >
    click me!
  </button>

</body> </html>
```

DOM-properties.html

Après avoir cliqué sur le bouton la valeur `"/.../mon_image.jpg"` est affichée sur une fenêtre d'alerte

Javascript et le DOM HTML

- Exemple 3 : (Javascript pur)
  - propriété `innerHTML` :
    le contenu HTML de l'élément (en tant que chaîne de caractères)

```
<!DOCTYPE html>
<html> <body>
  <div id="toto" >
    <p> Contenu original </p>
  </div>

  <script>
  var elem = document. getElementById("toto");
  elem.innerHTML ="<p> Nouveau contenu </p>"
  </script>

</body> </html>
```

innerHTML.html

Après l'exécution du script, l'HTML est modifié et...

Javascript et le DOM HTML

- Exemple 3 : (Javascript pur)
  - propriété `innerHTML` :
    le contenu HTML de l'élément (en tant que chaîne de caractères)

```
<!DOCTYPE html>
<html> <body>
  <div id="toto" >
    <p> Nouveau contenu </p>
  </div>

  <script>
  var elem = document. getElementById("toto");
  elem.innerHTML ="<p> Nouveau contenu </p>"
  </script>

</body> </html>
```

Voici le document affiché par le Navigateur

Javascript et le DOM HTML

Les variables déclarées dans les elements <script> sont globales

- Après le chargement de la page, les variables globales sont allouées en mémoire pendant toute la vie de la page
- Chaque variable globale est visible par le code exécuté après sa création

```
<!DOCTYPE html>
<html> <body>
  <img id= "img1"   src="mon_image.jpg" >

  <script>
  var elem = document. getElementById("img1");
  </script>
  <p> un paragraphe </p>

  <button onclick = "alert(str)" >
  click me!
  </button>

  <script>
  var str = elem.src;
  </script>
</body> </html>
```

visibilite.html

Javascript et le DOM HTML

- Remarque : Javascript peut insérer du Javascript dans une page!

  Le script suivant crée une fonction (anonyme) Javascript et l'affecte à l'attribut "onclick" du bouton "`btn`"

```
<!DOCTYPE html>
<html> <body>
  <button id="btn"> click me! </button>


  <script>
    var elem = document. getElementById("btn");
    elem.onclick = function(){alert(5+6);}
  </script>

</body> </html>
```

- On dit que le script ci-dessus associe un "*event handler*" au bouton "`btn`"

Javascript et le DOM HTML

- L' exemple précèdent à le même effet que :

```
<!DOCTYPE html>
<html> <body>

  <button id="btn" onclick="alert(5+6);"> click me!
  </button>

</body> </html>
```

- Avantage de créer un *event handler* dynamiquement : l'*event hanlder* peut dépendre du résultat de l'exécution d'un script

Javascript et le DOM HTML

- Plusieurs autres propriétés et méthodes disponibles dans le DOM!
- Grace au DOM HTML le code Javascript peut
  - parcourir l'arbre du DOM pour sélectionner des éléments HTML
  - lire /modifier
    - le contenu HTML d'un élément / le texte
    - la valeurs des attributs
    - le CSS
    - etc.
  - ajouter / éliminer des éléments HTML
  - etc.

Accès au DOM et librairies Javascript

- L'accès au DOM est possible en Javascript pur (voir exemples précédents)

- Cependant il a été grandement simplifié par des librairies, qui fournissent des méthodes de plus haut niveau pour
  - parcourir le DOM et sélectionner les éléments HTML
  - modifier l'HTML/CSS
  - en particulier : ajouter des *event handlers*

- JQuery est une de ces librairies les plus populaires
- On apprendra à manipuler le DOM et créer des *event handlers* directement en JQuery
- D'abord il faut connaitre les elements de base du langage Javascript

Plan

- Éléments du langage Javascript
  - les affectations
  - les opérateurs
  - les structure de contrôle
  - les tableaux
  - les objets
  - les fonctions
  - les variables globales

Éléments de la syntaxe de Javascript

- Déclaration de variable :

```
var x;    var x = expression;   var x = 3, y = 5, z = 1;
```

  - pas de déclaration de type, types dynamiques
  - `typeof x;` retourne le type de x

- Opérateurs arithmétiques : `+  -  *  /  %  ++  --`
  - Opérateurs logiques `&&  ||  !`
- Affectation : `x = "Roger";  x = y * z;  x += 3;  x *= 5;...`

- Chaines de caractères : `“Roger Rabbit”  ‘Roger Rabbit’`
- Concaténation de chaines de caractères: `+`

- Commentaires: `// commentaire  /* commentaire */`

Éléments de la syntaxe de Javascript

- Comparaisons :
  - `==` `!=` `>` `<` `>=` `<=`
  - valeurs et types égales : `===`
  - valeurs ou types différents : `!==`
  - `==` pour comparer les chaines de caractères
    (à condition que au moins une des chaines soit de type primitif, voir plus tard)

- Instructions : séparées par `;`
- Blocs d'instructions : `{ instructions; }`

- Structures de contrôle : `if-then-else, switch, for, while, do`
  - même syntaxe que Java (mais variables sans type)

Fonctions

```
function maFonction(a, b) {
    return a * b;
}
```

- Invocation de fonction : `maFonction(3, 5) ;`

- En Javascript les fonctions sont des objets
- Une déclaration de fonction est aussi une instruction qui crée un objet de type fonction
  - cet objet est implicitement affecté à une variable de nom égale au nom de la fonction (quand le nom de la fonction est spécifié)

Fonctions

- Manipuler un objet de type fonction :
  - Affectation : `var x = maFonction;`
  - Création et affectation d'une fonction anonyme :

```
var x = function (a, b) { return a * b; }
var y = x(2,3);
```

  - Passage comme argument :

```
monAutreFonction( maFonction );
monAutreFonction( x );
```

Objets : création

- Déclaration et création d'un objet :

```
var pers = {
    prenom :"Jean",
    nom : "Dupond",
    age: 50
};
```

  - Propriétés : `prenom, nom, age`
  - Méthodes : une méthode est une propriété avec une valeur de type fonction

```
var pers = {
  prenom :"Jean",
  nom : "Dupond",
  age: 50,
  nomComplet :
      function() { return this.prenom +' '+ this.nom; }
};
```

Objets : création

- Ajouter une propriété à un objet :

```
pers.nationalite = "France";
```

- Ajouter une méthode à un objet :

```
pers.changer =
   function (nouveauNom) { this.nom = nouveauNom; }
```

- Eliminer une propriété (ou méthode) d'un objet :

```
delete pers.age;
```

Objets : accès

- Accès aux propriétés d'un objet

  `pers.prenom` ou `pers["prenom"]` ou

  `pers[x];` où x a valeur `"prenom"`

- Accès aux méthodes :

```
var x = pers.nomComplet();
```

- **Attention** : `var x = pers.nomComplet` affecte à x un objet de type fonction

Objets : accès

- Une variante de la boucle `for` pour parcourir le propriétés d'un objet :

```
var pers = {prenom :"Jean", nom :"Dupond",age:50};
var x, str;
for (x in pers) {
    str += pers[x] + " ";
}
```

- Dans la boucle, `x` prends le valeurs "`prenom`" puis "`nom`" puis "`age`"

- Les objets Javascript sont tous adressés par référence

```
var pers = {prenom :"Jean", nom :"Dupond",age:50};
var z = pers;
```

`z` et `pers` font référence au même objet

Constructeurs

- En Javascript il n'y a pas de concept de classe
- Pour créer plusieurs objets avec les mêmes propriétés et méthodes on utilise une fonction de ce type, appelée constructeur:

```
function personne(p, n, a) {
     this.prenom = p;
     this.nom = n;
     this.age = a;
     this. nomComplet =
          function() { return this.prenom +' '+
                                   this.nom; }
}
```

- Création d'un objet avec le constructeur :

```
var x = new personne("Jean", "Dupond", 50);
```

- `new` crée un objet vide et le passe à la fonction `personne` qui le modifie (en lui ajoutant des propriétés et méthodes)

Constructeurs

- Un constructeur peut être appelé également sur un objet déjà existant :

```
var O={};
personne.call(O, "Jean", "Dupond", 50);
console.log(O.nomComplet());
```

Tableaux

- En Javascript les tableaux sont des objets
- Création d'un tableau : `var noms = ["Luc", "Jean", "Guy"];`
- Accès aux éléments d'un tableau : `t[0], t[1], ....`
- Un tableau Javascript peut contenir des valeurs de type diffèrent :

```
var t;
t[0] = 5; // un nombre
t[1] = maFonction; //une fonction
t[2] = noms; //un autre tableau
```

- Plusieurs propriétés et méthodes prédéfinies
  (i.e dans le prototype `Array.prototype`):

`t.length;  t.sort(); t.join("sep"); t.concat(tableau)`

etc.

Chaînes de caractères

- Les autre prototypes (`String, Number, Date` etc.) fournissent également un riche ensemble de propriétés et méthodes prédéfinies
- Quelques méthodes prédéfinies sur les chaînes de caractères
  - `str.charAt(0)` : caractère en position 0
  - `str.indexOf("hello")` : indice de la premiere occurrence de "hello" dans `str`
  - `str.substr(2,5)` extrait la sous-chaîne de `str` de taille 5 à partir de l'indice 2
  - `str.valueOf()` renvoie la valeur primitive de `str` (si `str` est un objet)
  - `str.trim()` enlève les espaces des deux extremes de `str`

- Pour une liste exhaustive consulter la référence Javascript!

Visibilité et temps de vie des variables

- **Variables locales :**
  - tous les paramètres des fonctions et les variables déclarées dans les fonctions

```
function maFonction(a, b) {
   var x = a*b; return x;
}
```

`a, b, x :` variables locales

  - visibles uniquement dans la définition de la fonction
  - créés quand la fonction est invoquée (ou à leur déclaration)
  - détruites quand la fonction termine

- toutes les variables déclarées dans les attributs HTML

```
<button onclick = "var s=5; alert(s)" >
 click me !
</button>
```

Visibilité et temps de vie des variables

- **Variables globales :** variables déclarées en dehors d'une fonction ou attribut

```
<script>
var x=5;
...
</script>
```

  - créés à au chargement de l'element <script> qui les contient (au moment de leur déclaration)
  - détruites quand la page Web est fermée
  - visibles par toutes les fonctions est scripts de la page qui sont exécutés après leur creation

Où inclure le Javascript coté client

- Dans le `head` ou dans le `body`
- Mais attention : le Javascript délimité par `<script> </script>` sera exécuté pendant le chargement de l'élément `script` correspondant...
- ...et les éléments HTML sont chargés dans l'ordre du document
- Conséquence : quand le script ci-dessous est exécuté, l'élément `"btn"` n'existe pas encore

```
<!DOCTYPE html>
<html> <body>

  <script>
    document.getElementById("btn").onclick =function(){alert(5+6);}
  </script>

  <button id="btn"> click me! </button>

</body> </html>
```

- l'*event handler* ne sera pas associé au bouton, l'effet désiré n'est pas obtenu!

Où inclure le Javascript coté client : une solution

Inclure les scripts toujours en fin de document (dernier élément du body)

```
<!DOCTYPE html>
<html> <body>

  <button id="btn"> click me! </button>

  <script>
    document.getElementById("btn").onclick =function(){alert(5+6);}
  </script>


</body> </html>
```

ou

```
<!DOCTYPE html>
<html> <body>

  <button id="btn"> click me! </button>

  <script src="mon-script.js"> </script>

</body> </html>
```

Où inclure le Javascript coté client : une alternative?

- Une autre possibilité , inclure un seul script :

```
<script>

 window.onload =
                function(){tout le javascript associé à la page}
</script>
```

- Ce script :
  - associe un *event handler* à l'événement “fenêtre complètement chargée”
  - peut être positionné n’importe où, par exemple dans le `head`
- l’*event handler*, et donc tout le Javascript de la page, sera déclenché avec le chargement complet de la page

- Cette solution permet d’inclure le Javascript dans le `head` (voir prochain slide)

Où inclure le Javascript coté client : une alternative?

```
<!DOCTYPE html>
<html>
<head>

<script>
window.onload = function(){
document.getElementById("btn").onclick =function(){alert(5+6);}
}
</script>

</head>

<body>
  <button id="btn"> click me! </button>

</body> </html>
```

window-onload.html

Ce code crée maintenant correctement l'*event handler* pour le bouton, une fois la page chargée

Où inclure le Javascript coté client : une alternative?

- Inconvénient de cette alternative :

  - l'événement `onload` est déclenché quand toute la page est chargée (y compris les images etc...)

  - en général les scripts ont juste besoin que le DOM soit complètement chargé, ce qui arrive plus tôt

  - la librairie JQuery offre une variante de cette solution, mais plus efficace

JQuery

quelques références et tutoriels utiles

http://www.w3schools.com/jquery/

http://jquery.com/

Introduction à JQuery

- JQuery : une librairie Javascript
- En pratique :
  un fichier .js qui sera stocké sur le serveur et attaché aux pages HTML comme tout autre script Javascript:

```
<head>
<script src="jquery-3.3.1.js"></script>
</head>
```

  télécharger `jquery-3.3.1.js` : http://jquery.com/download/

- Alternative : sans télécharger le fichier, spécifier une adresse où il est disponible

```
<head>
<script src="https://code.jquery.com/jquery-3.3.1.js">
</script>
</head>
```

Introduction à JQuery

- En JQuery l'opérateur `$( )` transforme un objet Javascript en un objet JQuery
- Des nouvelles méthodes, de plus haut niveau, sont alors disponibles sur l'objet
- Objets auxquels `$( )` est applicable : tous les objets Javascript qui représentent les éléments du DOM HTML

- Attention :

  les méthodes de l'objet Javascript `obj` (e.g. `innerHTML`, `getElementById(),…)` ne sont plus applicables à `$(obj)`

JQuery : syntaxe de base

```
$(sélecteur).action()
```

- `$(sélecteur)`: sélectionne des éléments dans le document HTML

  `sélecteur :` e.g. un sélecteur CSS : `"#monid", "p", ".maclasse"` etc.

  - `action()` : une action JQuery à exécuter sur tous les éléments sélectionnés
    - E.g. `hide()` cache l'élément (i.e. ajoute `display:none` à son style)

- **Exemples :**
  - `$("p").hide()` : cache tous les éléments <p> dans le document
  - `$("div.maclasse").hide()` : cache tous les éléments de classe `maclasse` dans le document

Exemple complet

```
<!DOCTYPE html>
<html>
<head> <script src="https://code.jquery.com/jquery-3.3.1.js">
</script> </head>
<body>

<h2> Un titre </h2>
<p> Un paragraphe </p>
<p class= "maclasse" > Un autre paragraphe </p>


<script>

   $("p.maclasse").hide();

</script>

</body>
</html>
```

Résultat : le deuxième paragraphe ne sera pas affiché

JQuery : syntaxe de base

```
$(sélecteur).action(fonction)
```

- `action (fonction)`: attache la fonction *`fonction`* à l'événement spécifié par `action`, sur tous les éléments sélectionnés

Exemples :

- `$("#btn").click(maFonction);`

  attache l'*event handler* `maFonction` à l'événement de click sur l'élément `#btn`

  Remarque : même effet que
  `document.getElementById("btn").onclick = maFonction;`

- `$("div.blue").mouseover(maFonction);`

  pour chaque `<div>` de classe `blue`, attache l'*event handler* `maFonction` au survol de la souris sur la `<div>`

Un exemple complet

```
<!DOCTYPE html>
<html>
<head> <script src="https://code.jquery.com/jquery-3.3.1.js">
</script> </head>
<body>

<h2>Un titre </h2>
<p>Un paragraphe</p>
<p>Un autre paragraphe</p>

<button id= "btn">Clique moi!</button>
<div style="margin-top:10pt"> survole moi! </div>

<script>
   $("#btn").click(  function(){ $("p").hide(); }   );

   $("div").mouseover(   function(){ $("p").show(); }   );
</script>

</body>
</html>
```

jquery-hide.html

Un exemple complet

Explication :

```
<script>

   $("#btn").click(   function(){ $("p").hide(); }    );

   $("div").mouseover(    function(){ $("p").show(); }    );
</script>
```

- Ce script associe
  - au click sur le bouton “btn”: une action qui cache tous les éléments <p> de la page
  - au survol de la seule du document : une action qui montre tous les éléments <p> de la page

Sélecteurs JQuery

Dans l'expression `$(sélecteur)`,

`sélecteur` peut être :

- Un sélecteur CSS
  - e.g. `"#monid"`, `"p"`, `"div"`, `"div.maclasse"`, `".maclasse"`, `"#div1>p"` etc..
  - l'ensemble des sélecteurs CSS a été étendu par JQuery (cf. références JQuery)

- Un objet Javascript représentant un élément du DOM
  - e.g. `document, this` (`this` sélectionne l'élément courant)

L'action `ready()`

`$(document).ready(`*`fonction`*`)`

associe la fonction *`fonction`* à l'événement "DOM du document chargé"

- La bonne solution pour s'assurer que tout le code Javascript/JQuery soit exécuté après le chargement du DOM :
  - Inclure seul le script suivant dans la page (par exemple dans le `head`)

```
<script>
$(document).ready( function(){
   // tous le code Javascript/JQuery de la page
});
</script>
```

- Cette solution n'a pas l'inconvénient d'attendre le chargement de toute la page
- et permet d'inclure le code Javascript/JQuery dans le `head`

L'action `ready( )`

Exemple

```
<!DOCTYPE html>
<html>
<head>
<script src="https://code.jquery.com/jquery-3.3.1.js"></script>

<script>
$(document).ready( function(){
   $("p.maclasse").hide();
});
</script>

</head>

<body>
<h2> Un titre </h2>
<p> Un paragraphe </p>
<p class= "maclasse" > Un autre paragraphe </p>
</body>
</html>
```

Actions JQuery

- Dans l'expression `$(sélecteur).action([fonction])`

  Un ensemble très riche de possibilités pour `action`

- On pourrait les classifier comme suit:
  - Événements : `onclick, mouseover, mouseleave,` ...
  - Actions d'effet / animation : `hide/show , slide, animate`, ...
  - Actions de manipulation HTML / CSS : `html, val, css, append`,...
  - Action de parcours du DOM : `parent, children, find`,...

- On en décrit quelques unes. Voir les références JQuery pour une liste exhaustive!!

Quelques événements JQuery

- Deja rencontrés : `ready, onclick, mouseover`
- Événements associés aux formulaires :

`$(selecteur).submit (`*`fonction`*`)`

  - associe l'exécution de *`fonction`* à la soumission du/des formulaires sélectionnés

`$(selecteur).change (`*`fonction`*`)`

  - associe l'exécution de *`fonction`* à un changement de contenu du/des champs de formulaire sélectionnés

`$(selecteur).focus (`*`fonction`*`)`

  - associe l'exécution de *`fonction`* au focus sur le/les champs de formulaire sélectionnés

`$(selecteur).blur (`*`fonction`*`)`

  - associe l'exécution de *`fonction`* à la perte de focus sur le/les champs de formulaire sélectionnés

Attacher des *event handlers* avec on( )

- Une méthode générique pour associer des événements aux éléments :

plusieurs événements :

```
$(selecteur).on ({
   click : fonction1,
   mouseenter: fonction2,
   mouseleave: fonction3,
   ...
});
```

un seul événement :

```
$(selecteur).on (
"click", fonction
);
```

- Remarque : plusieurs *event handlers* peuvent être associés au même élément **pour le même événement** :

```
$("p").click (fonction1);
$("p").click(fonction2);
$("p").on(“click”, fonction3);
```

Eliminer des *event handlers* avec `off()`

- Éliminer tous les *event handlers* associés à un événement

```
$(selecteur).off("event");
```

- Éliminer un *event handler* en particulier :

```
$(selecteur).off (
"event", fonction
);
```

Quelques effets / animations JQuery

- Déjà rencontrés : `hide / show`
- `$(selector).slideDown();  $(selector).slideUp();`
  - provoque le déroulement vers le bas (haut) du/des éléments sélectionnées (effet type menu déroulant)
- `$(selector).animate ({`*`du CSS avec syntaxe JQuery`*`})`
  - ajoute le CSS passé en argument au style du/des éléments sélectionnés
  - le changement de style est effectué progressivement, en créant un effet d'animation
  - seulement les propriétés CSS avec valeurs numériques peuvent être spécifiées
  - la syntaxe JQuery pour CSS est un peu différente
    - propriétés séparées par virgule
    - valeurs entourées par ' '
    - notation "*camel*" : `marginLeft` (au lieu de `margin-left`)

Un exemple d'animation JQuery

```
<!DOCTYPE html>
<html>  <head>
<script src="https://code.jquery.com/jquery-3.3.1.js"></script>
<script>
$(document).ready( function(){
    $("button").click( function(){
        $("div").animate({
            left: '250px',
            height: '150px',
            width: '150px',
            borderWidth: '5px'});
    });
});
</script> </head>

<body>
<button>Lancer l'animation</button>
<div style="background-color:blue; height:100px; width:100px;
position:absolute; border:solid 0px"></div>
</body></html>
```

jquery-animate.html

Callbacks

- Tous les effets qui ont une durée prennent comme paramètre optionnel une fonction
  - cette fonction, dite, de "callback" sera exécutée à la fin de l'animation

Exemple

```
$("div").animate({…}, function(){
        $("#par1").text("animation finie!");
});
```

query-callback.html

Quelques méthodes JQuery de manipulation HTML

- `$(selecteur).html()` / `$(selecteur).html("du html")`
  - renvoie / change l'html contenu dans le premier élément sélectionné
- `$(selecteur).replaceWith("du html")`
  - remplace chacun des éléments sélectionnés avec l'html en argument
- `$(selecteur).attr("nom d'attribut")` / `$(selecteur).attr("nom d'attribut", "du texte")`
  - renvoie / change la valeur de l'attribut spécifié des éléments sélectionnés
- `$(selecteur).val()` / `$(selecteur).val("valeur")`
  - renvoie / change la valeur des éléments d'input (<input>) sélectionnés
- ...

Quelques méthodes JQuery de manipulation CSS

- `$(selecteur).css("nom de propriété")`
  - renvoie la valeur de la propriété CSS spécifiée, pour le premier élément sélectionné

- `$(selecteur).css(`
  `{"nom de propriété":"valeur",`
  `"nom de propriété": "valeur",...});`
  - modifie la valeur des propriétés CSS spécifiées, pour tous les éléments sélectionnés
  - remarque : syntaxe CSS classique pour les noms de propriété

- ...

Quelques méthodes JQuery pour parcourir le DOM

- `$(selecteur).parent()/parents()/children()/find()/...`
  - sélectionne les parents / ancêtres / enfants / descendants /... des éléments initialement sélectionnés
    - Ex. `$(#id1).children().hide()`

      cache les elements enfants de l'element `#id1`
  - argument optionnel : un sélecteur css pour filtrer ultérieurement la sélection
    - `$(#id1).children(".maclasse").hide()`

      cache les enfants de l'element `#id1` qui sont de classe `maclasse`

Quelques méthodes JQuery pour parcourir le DOM

```
<!DOCTYPE html>
<html>  <head>
<script src="https://code.jquery.com/jquery-3.3.1.js"></
script>
<script>
$(document).ready( function(){
   $("button").click(function(){
      $("#id1").children(".maclasse").hide();
   });
});
</script>
</head>
<body>
<ul id=“id1">
   <li> un </li>
   <li class="maclasse"> deux </li>
   <li> trois </li>
</ul>
<button> click me </button>
</body> </html>
```

jquery-dom.html

Quelques méthodes JQuery pour parcourir le DOM

- `$(selecteur).first() / last()`
  - réduit la sélection au premier / dernier élément de la sélection initiale
- `$(selecteur).eq(`*`i`*`)`
  - réduit la sélection à l'élément numero *i* de la sélection initiale (les index commencent à 0)
- `$(selecteur).filter("sélecteur CSS")`
  - réduit la sélection initiale aux éléments qui satisfont le sélecteur en argument
- `$(selecteur).length` renvoie le nombre d'elements dans la selection

Chaînage d'actions JQuery

- Les actions JQuery peuvent être enchainées sur la même sélection :

Exemple

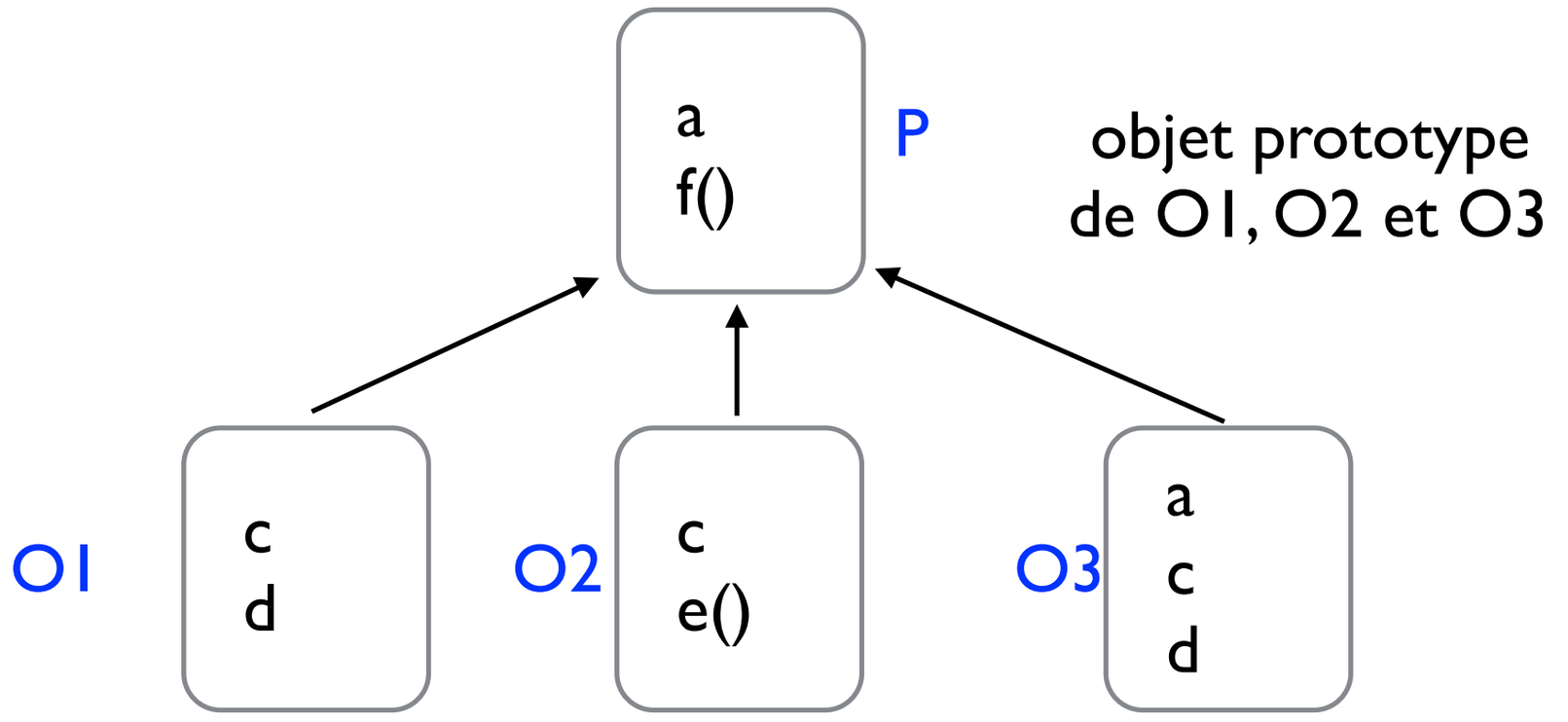
```
$("#monid").css("color", "red").slideUp().slideDown();
```

- Les différentes actions sont exécutes dans l'ordre

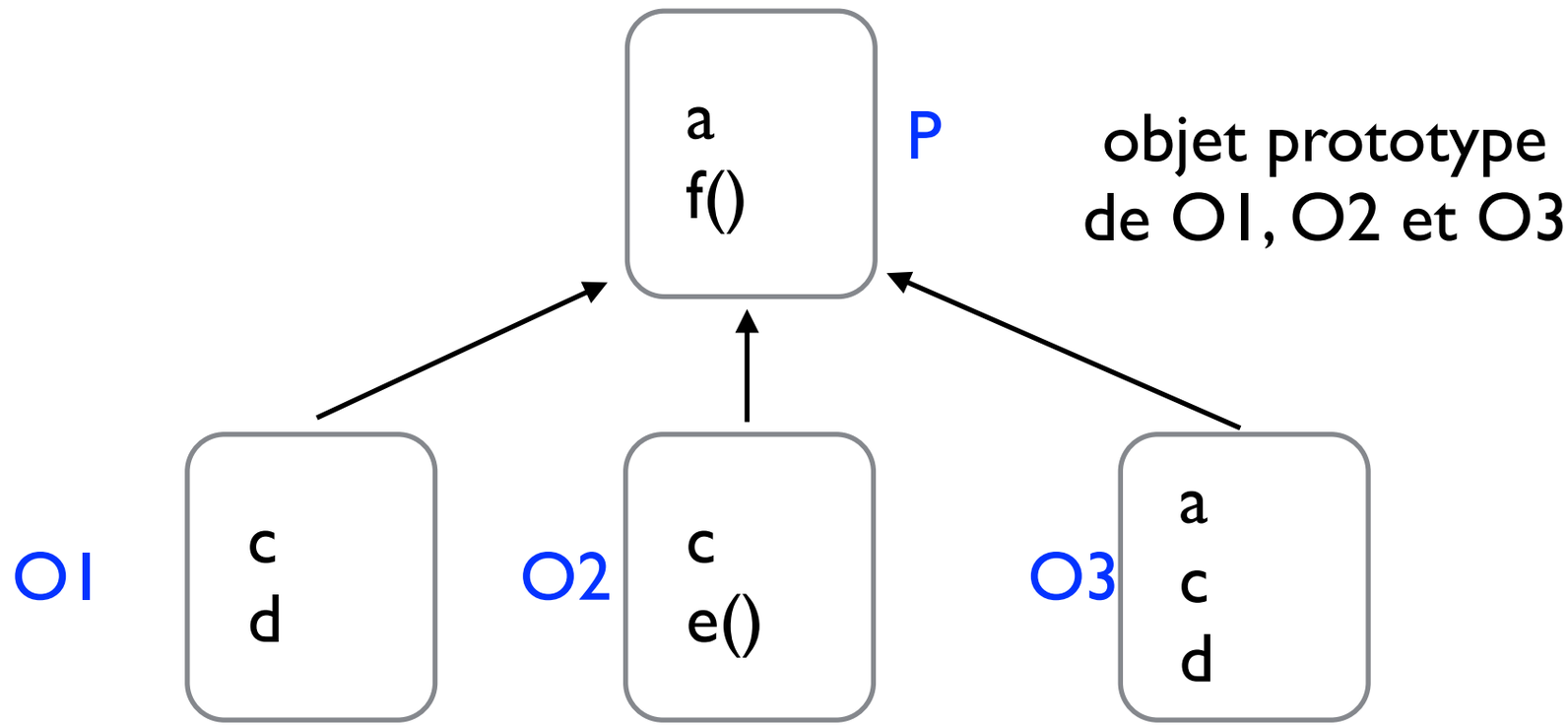
Compléments de Javascript

Prototypes

- À différence de Java, C++, ... qui sont des langages “de classes”,
  Javascript est un langage de prototypes
- **Chaque objet a un prototype** (sauf l’objet null).
  un prototype est un autre objet, et plusieurs objets peuvent partager le même prototype



Prototypes



- Les propriétés (et méthodes) du prototype sont accessibles depuis l'objet (héritage) :
  - propriétés de O1 : O1.c, O1.d (propres)
    O1.a (= P.a) , O1.f() (= P.f()) (héritées)

- Si un objet redéfinit une propriété avec le même nom qu'une propriété de son prototype, la propriété locale de l'objet "cache" celle du prototype (*overriding*)
  - Exemple : O3.a fait reference à la propriété locale de O3, et non pas à P.a

Prototypes : rattachement

- Le prototype d'un objet est rattaché à celui-là au moment de sa création
- Toute fonction `f()` en Javascript est associée à un objet (initialement vide) accessible comme `f.prototype`
- Tout objet crée avec `new f()` aura comme prototype `f.prototype`

```
function Compte(montant, proprietaire) {
 this.montant = montant;
 this.proprietaire = proprietaire;
}
var o = new Compte(1000, "Cristina");
// o a deux propriétés propres
   (o.montant, o.proprietaire)
// et est rattaché à un prototype vide
var m = o. montant;// m == 1000
var p = Compte.prototype;
var t = typeof (p); // t == “object”
var x = Object.keys(p).length; // x == 0
var y = Object.keys(o).length; // y == 2
```

Prototypes : rattachement

- Un objet peut être créé et rattaché explicitement à un autre objet comme prototype:

```
var P = {a: 1};

var O = Object.create(P);

// crée O et le rattache au prototype P

console.log(O.a); // 1 (hérité de P)
```

Prototypes : modification

- Des propriétés peuvent être ajoutées à tout moment à un prototype
- On met typiquement dans le prototype des propriétés qui ne dépendent pas de l'instance particulière

```
function Compte(montant, proprietaire) {
 this.montant = montant;
 this.proprietaire = proprietaire;
}
var o = new Compte(1000, "Cristina");
var o1 = new Compte(100, "Sylvain");
// o et o1 ici sont rattachés au même prototype vide
Compte.prototype.E2D = 1.074; //euros vers dollars
Compte.prototype.montantD = function() {
                return this.montant * this.E2D;}

// o et o1 sont rattachés ici au même prototype avec 2
propriétés
…
```

Prototypes : modification

```
function Compte(montant, proprietaire) {
 this.montant = montant;
 this.proprietaire = proprietaire;
}
var o = new Compte(1000, "Cristina");
var o1 = new Compte(100, "Sylvain");
// o et o1 ici sont rattachés au même prototype vide
Compte.prototype.E2D = 1.074; //euros vers dollars
Compte.prototype.montantD = function() {
                return this.montant * this.E2D;}

// o et o1 sont rattachés ici au même prototype avec 2
propriétés
var x = Object.keys(Compte.prototype).length; // x == 2
var e = o.E2D;  // e == 1.074
var e1 = o1.E2D;  // e1 == 1.074
var m = o.montant; // m == 1000
var d = o.montantD(); // d == 1074
var d1 = o1.montantD(); // d1 == 170.4
```

Prototypes : modification partagée

```
function Compte(montant, proprietaire) {
 this.montant = montant;
 this.proprietaire = proprietaire;
}
var o = new Compte(1000, "Cristina");
var o1 = new Compte(100, "Sylvain");
// o et o1 ici sont rattachés au même prototype vide
Compte.prototype.E2D = 1.074; //euros vers dollars
Compte.prototype.montantD = function() {
                  return this.montant * this.E2D;}

var e = o.E2D;  // e == 1.074
var e1 = o1.E2D;  // e1 == 1.074
//les changements dans le prototype sont visibles dans
chaque objet
Compte.prototype.E2D = 1.075;
e = o.E2D;  // e == 1.075
e1 = o1.E2D;  // e1 == 1.075
```

Prototypes : modification locale

- Attention : une propriété du prototype ne peut pas être modifiée depuis un objet ayant ce prototype :
  - `o. E2D = 1.080;`
    est interprété comme l'ajout d'une nouvelle propriété propre à o, qui fait *overriding* de la propriété correspondante du prototype

Prototypes : modification locale

```
function Compte(montant, proprietaire) {
 this.montant = montant;
 this.proprietaire = proprietaire;
}
Compte.prototype.E2D = 1.074; //euros vers dollars
Compte.prototype.montantD = function() {
                  return this.montant * this.E2D;}
var o = new Compte(1000, "Cristina");
var o1 = new Compte(100, "Sylvain");

o1.E2D = 1.080;
//o1 a une nouvelle propriété propre E2D
var e = o.E2D;  // e == 1.074
var e1 = o1.E2D;  // e1 == 1.080
var m = o.montantD(); // m == 1074
var m1 = o1.montantD(); // m1 == 108
```

prototypes.html

Constructeurs et prototypes natifs

- En Javascript toutes les valeurs sont des objets à exception des valeurs primitives :
  - chaînes de caractères ("Jean Dupond"),
  - nombres (3.14),
  - booléens (true, false)

- Il existe un constructeur pre-défini (et prototype associé, dit natif) pour chaque type non-primitif du langage :

  `Object(), Array(), Function(), Date() etc..`

  toutefois créer les objets par affectation de littéraux est plus efficace

- Il existe également la version à objets des types primitifs

  `String() Number() Boolean ()`

  mais l'utilisation de types primitifs est à préférer

Constructeurs et prototypes natifs

- Un objet déclaré par affectation d'un littéral est créé par invocation implicite du constructeur natif correspondant
- Donc

```
var pers = {
  prenom :"Jean",
  nom : "Dupond",
  age: 50,
};
```

est équivalent à

```
var pers = new Object();
pers.prenom :"Jean";
pers.nom : "Dupond";
pers.age: 50;
```

`pers` est donc rattaché au prototype Object.prototype