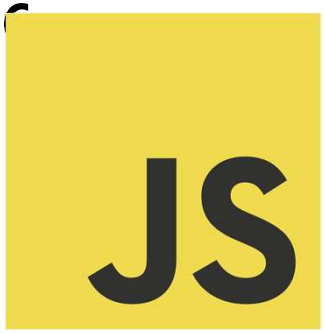
JavaScript ?

- Langage créé en **1995**
- Issu d'un langage de développement coté serveur (LiveScript)
- Un partenariat Netscape/Sun donnera son nom définitif : **JavaScript**
- Soumis a L'**Ecma** (organisme de standardisation) en **199**
- JavasScript devient un langage à part entière **1997**



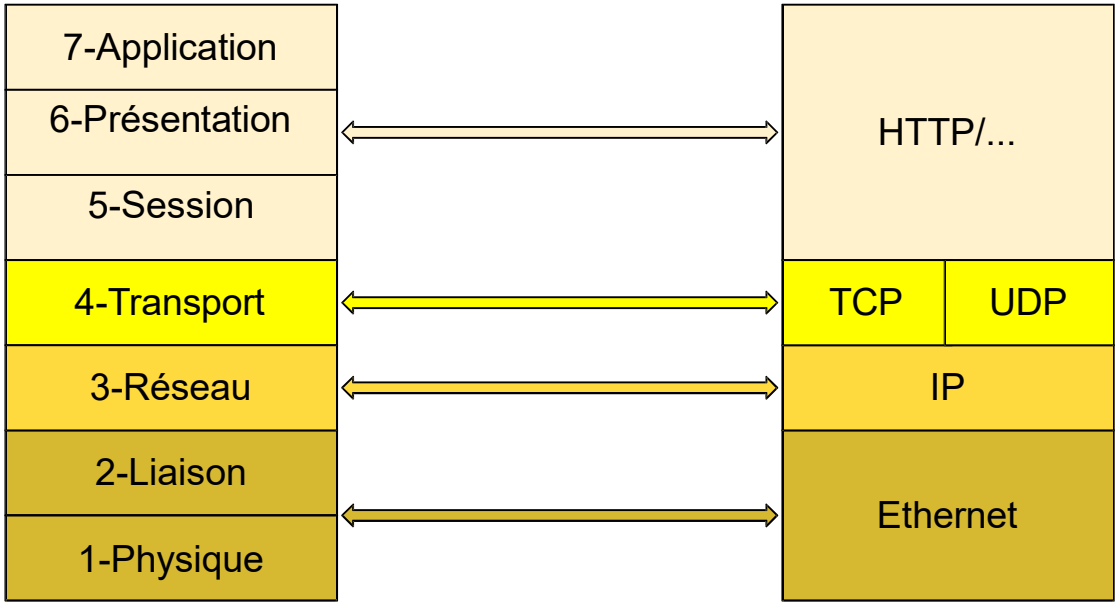
- **Concrètement :**
  - JavaScript va nous permettre de manipuler une page HTML directement dans le navigateur Web de l'utilisateur

Le plan de formation

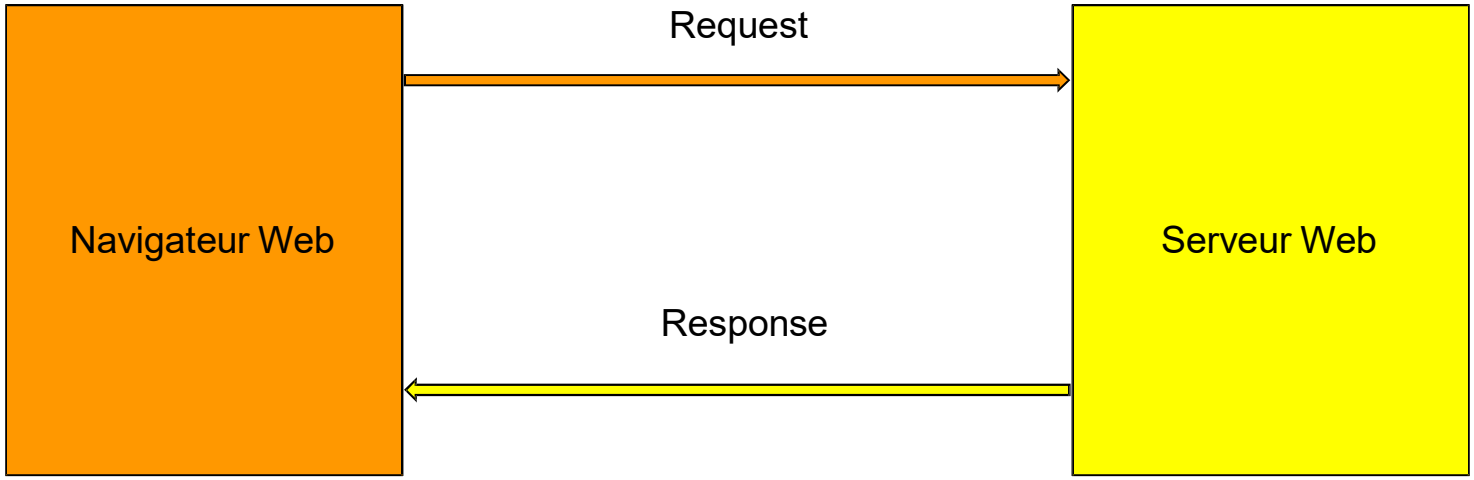
1. Introduction au développement Web
2. Les bases du langage
3. Les pièges du langage
4. Introduction à la programmation orienté objet
5. Javascript dans le navigateur
6. Javascript et la communication avec le serveur

Le protocole HTTP

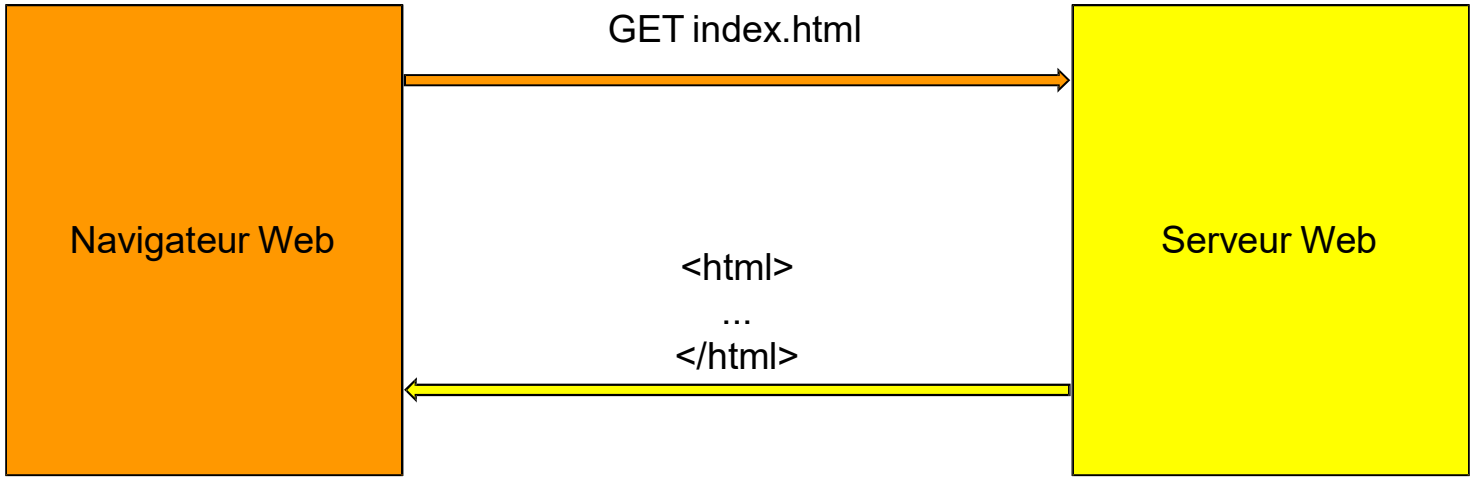
Modèle OSI



Le protocole HTTP : Un échange HTTP 1/2



Le protocole HTTP : Un échange HTTP 2/2



Le protocole HTTP : Les méthodes

GET	**Demander une ressource**
POST	**Transmettre des données à une ressource**
PUT	**Remplacer ou ajouter une ressource**
DELETE	**Supprimer une ressource**
HEAD	Demander des informations sur la ressource
OPTIONS	Connaître les options de communications d'une ressource
CONNECT	utiliser un proxy comme tunnel de communication
TRACE	permet d'effectuer des tests
PATCH	modification partielle d'une ressource

HelloWorld

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    alert("HelloWorld");
    console.log("HelloWorld");
  </script>
</head>
<body>

</body>
</html>
```

Plan

- Notion de variable
- Déclarer une variable
- Initialiser une variable
- Les contraintes de nommage
- Les opérateurs
- Les types primitifs et l'opérateur **typeof**

Notion de variable

- Utilisées pour stocker des valeurs
- Deux étapes pour les utiliser :
  - Déclarer la variable
  - Initialiser la variable

Déclarer une variable

- L'instruction **var**
- L'instruction **let**
- L'instruction **const**

Initialiser une variable

- Il y a 2 possibilités :
  - Déclarer la variable et l'initialiser en 2 étapes :

```
var a; // déclaration
a=1; // initialisation
```

  - Déclarer la variable et l'initialiser en même temps :

```
var a = 1; // déclaration et initialisation
```

Les contraintes de nommages

- Les variables sont “**case sensitive**”
- C’est à dire :

```
var a = “minuscule”;
var A = “MAJUSCULE”;
```

  - **a et A sont deux variables différentes !**
- Il est recommandé d’utiliser le plus possible la touche de “complétion” de votre éditeur (touche TAB).
- Elle doivent commencer par une lettre

Les opérateurs

- Les opérateurs arithmétiques

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
++	Incrément
--	Décrément

Les types de variables et l'opérateur typeof

- En JavaScript les variables sont **typées**
- Il y a **5** types primitifs
  - **Number :** entier ou flottant
  - **String :** chaine de caractères
  - **Boolean :** (true ou false)
  - **Undefined :** quand la variable que vous souhaitez utiliser n'existe pas
  - **Null :** type particulier n'ayant q'une seule valeur : **null**
- Comment connaître le type d'une variable : l'opérateur **typeof**.

Plan

- Notion de tableau
- Déclarer et initialiser un tableau
- Accéder aux éléments d'un tableau
- Les opérations sur les tableaux

Notion de tableau

- Un tableau est **une variable** qui contient une **liste de valeur**
- Représentation d'un tableau :

0	Une valeur
1	Une autre valeur
2	Encore une autre valeur

Déclarer et initialiser un tableau

- Déclaration d'un tableau vide

```
var a = [];
```

- Déclaration d'un tableau avec des valeurs

```
var a = [1,2,3];
```

Accéder aux éléments d'un tableau

- Les éléments d'un tableau sont accessibles par un index.
- Cet index est un nombre entier avec pour valeur initiale **0**.

- En reprenant cet exemple :

```
var a = [1,2,3];
```

- On en déduit la structure du tableau et la façon d'accéder aux éléments :

0	1
1	2
2	3

```
console.log(a[0]);
console.log(a[1]);
console.log(a[2]);
```

Modifier ou Ajouter une valeur

- Modification de la valeur située à l'index **2**.

```
a[2] = "la valeur 3";
```

- Ajout d'une valeur à l'index **3**.

```
a[3] = "la valeur 4";
```

- Ajout d'une valeur à l'index **6 (!)**

```
a[6] = "la valeur 7";
```

Contenu d'un tableau

- Un tableau peut contenir différents types de données...

```
var a =[1,"deux",false,null,undefined];
```

- ... même d'autres tableaux

```
var a =[1,"deux",false,null,[1,2,3]];
```

- Pour accéder au tableau dans le tableau :

```
console.log(a[4][1]);
```

Plan

- Notion de bloc de code
- Structure d'une condition if
- Structure d'une condition if - else
- Structure d'une condition if - else if - else
- La notation ternaire
- Le Switch

Notion de bloc de code

- Un bloc de code est un ensemble d'instructions encadré par des accolades

```
{
  var a = 1;
  var b = 2;
}
```

- Tout le code du bloc est exécuté au moment où le bloc est sollicité

Notion de comparaison

- Une comparaison renvoie toujours une valeur **booléenne**
- Elle est constituée :
  - d'un opérateur de comparaison
  - d'un **opérateur logique**
  - de n'importe quelle **valeur** pouvant être convertie en booléen

Les opérateurs

- Logiques :
  - !(NOT)
  - &&(AND)
  - || (OR)
- Comparaisons :
  - == (EQUAL)
  - === (STRICT EQUALITY)
  - < (LESS THAN)
  - > (GREATER THAN),…

Structure d’une condition if

```
var a = 2;
var result = '';
if(a>1){
  result = "a est supérieur à 1";
}
```

Structure d’une condition if - else

```
var a = 1;
var result = '';
if (a > 2) {
  result = "a est supérieur à 2";
} else {
  result = "a n'est pas supérieur à 2";
}
```

Structure d’une condition if - else if - else

```
var a = 1;
var result = '';

if(a==1){
  result = "a==1";
  }
else if(a==2){
  result = "a==2";
}
else if(a==3){
  result = "a==3";
}
else {
  result = " a est différent de 1,2,3";
}
```

La notation ternaire

- Ce code peut être simplifié :

```
var a = 1;
var result = '';
if (a > 2) {
  result = "a est supérieur à 2";
} else {
  result = "a n'est pas supérieur à 2";
}
```

- Par :

```
//notation ternaire
var a = 1;
var result = a<2?"a est supérieur à 2":"a n'est pas supérieur à 2";
```

Le switch

```
var a = 1;
var result = '';
switch (a) {
  case 1:
    result = 'a = 1';
    break;
  case 2:
    result = 'a = 2';
    break;
  default:
    result = 'a n\'est différent de 1 et 2 ';
    break;
};
```

Plan

- La boucle while
- La boucle do…while
- La boucle for
- La boucle for…in
- La boucle for...of

La boucle while

```
var a = 0;
while (a < 10) {
  a++;
}
```

La boucle do…while

```
var a = 0;
do {
  a++;
} while (a < 10);
```

La boucle for

```
for (var a = 0; a < 10; a++) {
  console.log(a);
}
```

La boucle for…in

```
var a = ['un','deux','trois','quatre'];
for (var i in a) {
  console.log(i+' '+a[i]);
}
```

La boucle for…of

- Nouveauté ECMAScript6 !

```
var a = ['un','deux','trois','quatre'];
for (var b of a) {
  console.log(b);
}
```

Plan

- Notion de fonction
- Appeler des fonctions et passer des paramètres
- Quelques fonctions du langage
- Portée des variables
- Considérations importantes sur les fonctions

Notion de fonction

- Les fonctions sont des blocs de code réutilisables

```
function somme(a,b){
  var c = a+b;
  return c;
}
```

- Une fonction renvoie toujours une valeur (**return**)!

Appeler des fonctions et passer des paramètres 1/2

```
function somme(a,b){
  var c = a+b;
  return c;
}

var resultat = somme(2,3);
console.log(resultat); //5
```

2/2

- Tous les paramètres sont disponibles dans une variable spéciale : **arguments**

```
function une_function(){
  console.log(arguments);
}
une_function(1,"un");
```

Quelques fonctions du langage

- **parseInt()** : renvoie un entier à partir d'une chaine de caractères
- **parseFloat()** : renvoie un flottant à partir d'une chaine de caractères
- **isNaN()** : teste si le paramètre n'est pas un nombre
- **isFinite()** : teste si le paramètre n'est pas Infinity

Portée des variables

- La portée d'une variable **n'est pas** définie par un bloc de code !
- La portée d'une variable **est définie** par une fonction
- Une variable doit être déclarée avec **var**

```
var global = 'variable globale';

function une_function(){
    var local = 'variable locale';
}
```

```
var global = 'variable globale';

function une_function(){
    local = 'variable n\'est pas locale';
}
```

Considérations importantes sur les fonctions 1/2

- Les fonctions sont des données, elles sont typées !

```
function une_function(){
     console.log('une_function');
}

console.log(typeof une_function);
```

- On peut en déduire cette notation

```
var une_function = function(){
     console.log('une_function');
};

console.log(typeof une_function);
```

Considérations importantes sur les fonctions 2/2

- Les fonctions ont **3 caractéristiques :**
  1. Elles contiennent du **code**
  2. Elles sont **exécutables**
  3. Elles se comportent comme des **variables**

Les Exceptions

- Les exceptions sont la représentation d'un comportement exceptionnel
- Elles interrompent le flot d'exécution normal de votre application
- Elles évitent la scrutation d'une erreur

La console

- L'API console fournit par les navigateurs offre plus de possibilités que le simple console.log("…");
  - **console.info()** : affiche un message d'information

```
> console.info('un message d\'information')
un message d'information
```

  - **console.error()** : affiche une erreur

```
> console.error('un message d\'erreur');
▶ un message d'erreur
```

  - **console.warn()** : affiche un message de warning

```
> console.warn('un message de warning');
un message de warning
```

  - …

La structure try…catch

- Provoquons une erreur, ici la variable **ma_variable** n'existe pas.

```
console.log(ma_variable);
```

- Interceptons cette erreur

```
try{
  console.log(ma_variable);
}
catch(e){
  console.error("Il y a une erreur");
}
```

- Le bloc **try** exécute du code susceptible de provoquer une erreur
- Le bloc **catch** intercepte cette erreur si elle a lieu

La structure try…finally

- Le bloc **finally** est exécuté quelque soit le résultat de l'exécution du bloc try (erreur ou pas)

```
try{
  console.log(ma_variable);
}
finally{
  console.info("après le bloc try");
}
```

- Utile pour initialiser la valeur d'une variable pour la suite de l'exécution

La structure try…catch…finally

- C’est la forme la plus complète pour gérer des exceptions

```
try{
  console.log(ma_variable);
}
catch(e){
  console.error("Il y a une erreur");
}
finally{
  console.info("après le bloc try");
}
```

Lancer une exception avec throw

- Il est possible de lancer ses propres exceptions

```
function division(a,b){
  if(b===0){
    throw "Erreur : Division par zéro";
  }
  return a/b;
}

try{
  var c = division(2,0);
  console.log(c);
}
catch(erreur){
  console.log(erreur);
}
```

Qu’est ce que le hoisting

- En JavaScript, les déclarations de variables sont traitées avant tout !

```
a=2;
var a;

//est strictement équivalent à

var a;
a=2;
```

Le hoisting et la déclaration de variable

- **Rappel** : 2 étapes pour utiliser une variable :
  1. Déclaration
  2. Initialisation
- Nous le savons maintenant : les déclarations sont remontées (hoisted)

Le hoisting et l'initialisation de variable

- En revanche les initialisations, elles, ne sont pas remontées :

```
var a = 2; // Déclaration et initialisation de a
console.log(a+b);
var b = 3; // Déclaration et initialisation de b
```

- NaN ! b est bien déclarée (par hoisting) mais pas initialisé -> Undefined

Les objets ?

- D’après wikipedia :
  - « ... un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre. »
- Bref…
  - Les **types primitifs** sont des éléments purement informatiques : String, Number
  - Les **objets** des représentations du monde réel : une facture, un client, un utilisateur.

Les objets en JavaScript

- Proche de la notion de tableau :

```
var a = ['un','deux','trois'];
```

Key	Value
0	un
1	deux
2	trois

Les objets en JavaScript

```
var personne = {
  nom:"DURAND",
  prenom:"Robert"
};
```

Key	Value
nom	DURAND
prenom	Robert

Tableau indexé et tableau associatif

- Deux sortes de tableaux :
  1. **Indexé** -> les clefs sont des entiers
  2. **Associatif** -> les clefs sont des chaines de caractères
- JavaScript utilise les objets pour implémenter les tableaux associatifs

Structure d’un objet

- Les propriétés et les méthodes

```
var personne = {
  nom:"DURAND",
  prenom:"Robert",
  direBonjour : function(){
    console.log('Bonjour');
  }
};
```

Utilisation d’un objet

```
var personne = {
  nom:"DURAND",
  prenom:"Robert",
  direBonjour : function(){
    console.log('Bonjour');
  }
};

personne.direBonjour();
```

Création différé d'un objet

- JavaScript est un langage dynamique
- Il permet la modification d'un objet après sa création !

```
var personne={};
personne.nom = "DURAND";
personne.prenom = "Robert";
personne.direBonjour = function(){
  console.log('Bonjour');
};
```

La valeur this

- **this** : « cet objet » ou l'objet courant

```
var personne = {
  nom:"DURAND",
  prenom:"Robert",
  afficheNom : function(){
    console.log('nom : '+this.nom);
  }
};
```

- Ici on affiche la propriété **nom** de **l'objet courant**

Notion de constructeur 1/2

- Utiliser une function pourcréer un objet

```
function Personne(){
  this.nom = "DURAND";
  this.prenom = "Robert";
}

var personne = new Personne();
console.log(personne.nom);
```

Notion de constructeur 2/2

- Utiliser une **function** et passer des paramètres pour créer un objet

```
function Personne(p_nom,p_prenom){
  this.nom = p_nom;
  this.prenom = p_prenom;
}

var nom_personne = "DURAND";
var prenom_personne = "Robert";
var personne = new Personne(nom_personne,prenom_personne);
console.log(personne.nom);
```

L’opérateur instanceof

- Type primitif -> typeof
- Objet -> instanceof
- L’opérateur instanceof teste le constructeur de l’objet

```
function Personne(p_nom,p_prenom){
  this.nom = p_nom;
  this.prenom = p_prenom;
}

var nom_personne = "DURAND";
var prenom_personne = "Robert";
var personne = new Personne(nom_personne,prenom_personne);
console.log(personne.nom);


console.log(personne instanceof Personne ? "oui":"non");
```

L’objet Object

- **Object** est l’objet de base implémenté par l’ensemble des objets JavaScript.

```
var obj = {};
var obj = new Object();
```

- Autrement dit tous les objets JavaScript **héritent** de Object
- Ils bénéficient des méthodes de Object :
  - **toString()**
  - **valueOf()**

Qu’est ce qu’un prototype

- Rappel de **Wikipedia** :
  - Javascript est un langage **orienté objet à prototype**, c’est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une **propriété de prototypage** qui permet d’en créer des objets héritiers personnalisés.

Qu’est ce qu’un prototype

- On le sait, les functions sont des objets
- Elles ont des propriétés
- Une de ces propriétés est : **prototype**
- Il est possible d’ajouter des propriétés et des méthodes à partir du prototype

Ajouter des propriétés à un prototype

```
function Personne(p_nom,p_prenom){
  this.nom = p_nom;
  this.prenom = p_prenom;
}

Personne.prototype.age = 30;
Personne.prototype.getAge = function(){
  return this.age;
}

var personne = new Personne("DURAND","Robert");
console.log(personne.getAge());
```

Ajouter des propriétés à un prototype

- En Javascript les objets sont passés par référence.
- C'est-à-dire qu'il y a **un** prototype pour tous les objets du même constructeur.

```
var personne = new Personne("DURAND","Robert");
console.log(personne.getAge());
var personne2 = new Personne("DURAND","Robert");
console.log(personne2.getAge());
Personne.prototype.age = 130;
console.log(personne.getAge());
console.log(personne2.getAge());
```

prototype

- Dans notre exemple

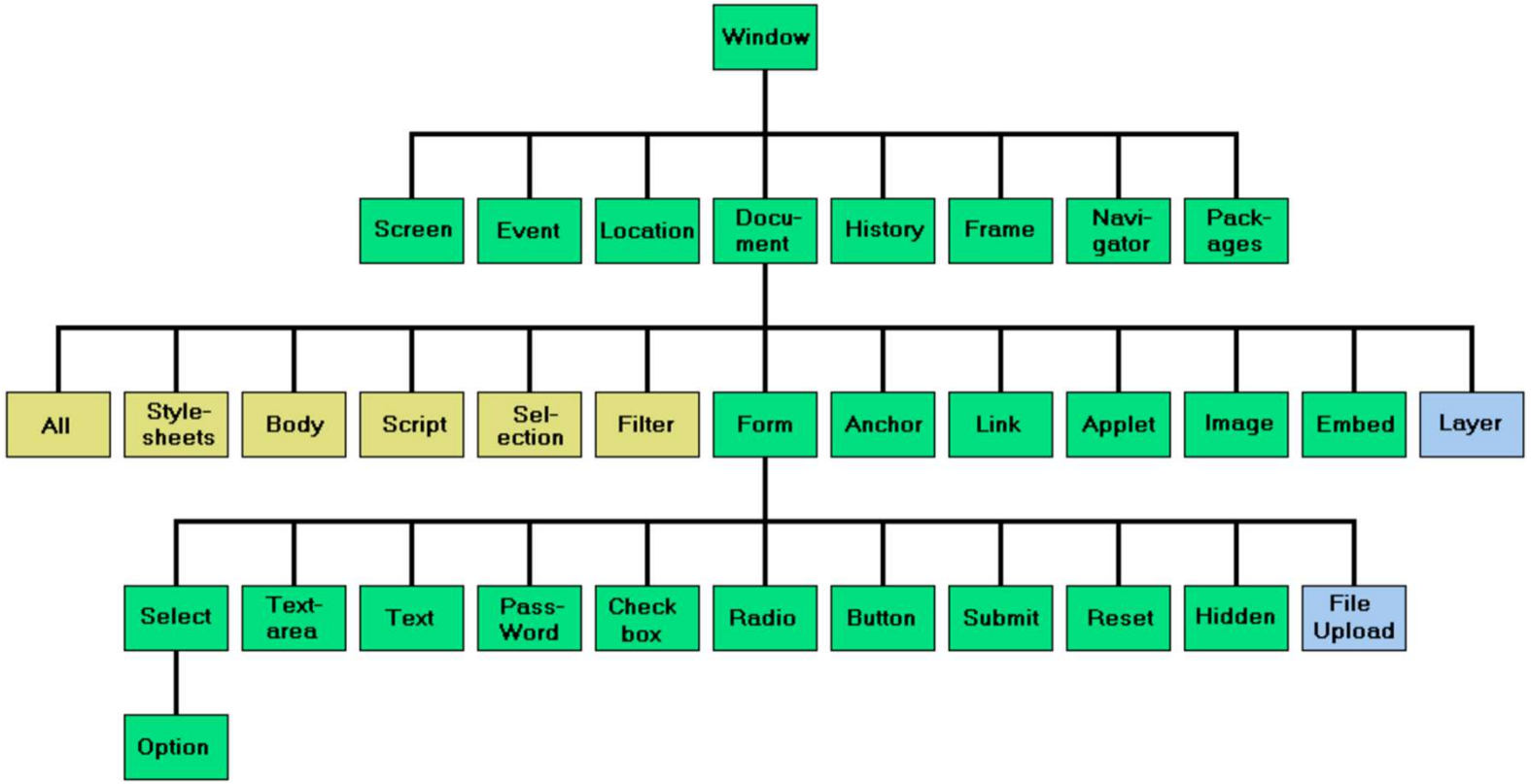
```
function Personne(p_nom,p_prenom){
  this.nom = p_nom;
  this.prenom = p_prenom;
}
Personne.prototype.age = 30;
var personne = new Personne("DURAND","Robert");
console.log(personne.age);
console.log(personne.constructor.prototype.age);
```

Améliorer les objets JavaScript (Array,…)

```
Array.prototype.showAll = function(){
  for(var i=0;i<this.length;i++){
      console.log(this[i]);
  }
};

var arr = ['un','deux','trois'];
arr.showAll();
```

DOM


Window
Screen
Event
Location
Docu-ment
History
Frame
Navi-gator
Pack-ages
All
Style-sheets
Body
Script
Sel-ection
Filter
Form
Anchor
Link
Applet
Image
Embed
Layer
Select
Text-area
Text
Pass-Word
Check box
Radio
Button
Submit
Reset
Hidden
File Upload
Option

L’objet Window

- L’objet **Window** représente la fenêtre du navigateur

```
console.log(window.innerWidth);
console.log(window.innerHeight);
```

L'objet **Form** et ses enfants

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    window.document.forms['le_form'].le_champ_text.value="Nouvelle valeur";
  </script>
</head>
<body>
  <form name="le_form" action="">
    <input type="text" name="le_champ_text">
  </form>
</body>
</html>
```

L’objet Document

- **Document** Object Model

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
  <h1 id="le_titre"></h1>
  <script>
    var le_titre = window.document.getElementById('le_titre');
    le_titre.innerHTML = "Un titre";
  </script>

</body>
</html>
```

Exécution du code JavaScript

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    alert('Bonjour');
  </script>
</head>
<body>
  <h1>Hello</h1>
</body>
</html>
```

Exécution du code JavaScript

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>

</head>
<body>
  <h1>Hello</h1>
  <script>
    alert('Bonjour');
  </script>
</body>
</html>
```

Analyses des 2 exemples

- Dans le premier exemple : l'alert bloque l'affichage du titre
- Dans le second exemple : le titre est affiché et l'alert s'affiche ensuite
- **Conclusion** le code d'une page Web est exécuté de façon séquentielle
- Or, nous souhaitons exécuter notre code quand la page est prête, c'est-à-dire quand elle est chargée.

Exécution au chargement de la page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
  function doload(){
    alert('hello');
  }
  </script>
</head>
<body onload='doload()'>
  <h1>Hello</h1>
</body>
</html>
```

Notion d’évènement

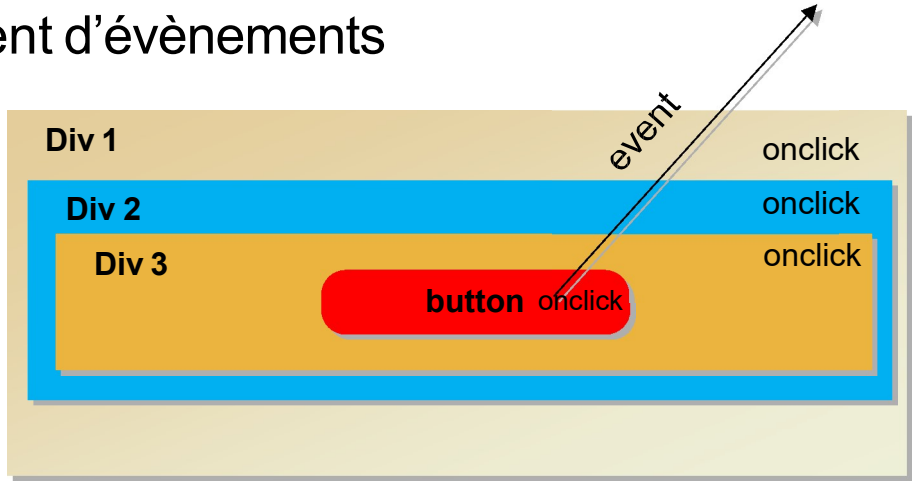
- Les évènements vont permettre de réagir à des actions faites par l’utilisateur ou des évolutions de la page.

Le principaux évènements

Event	Description
onchange	Un élément HTML a changé
onclick	Un utilisateur clique sur un élément HTML
onmouseover	Un utilisateur survol un élément HTML avec la souris
onmouseout	Un utilisateur quitte un élément HTML avec la souris
onkeydown	Un utilisateur appui sur un touche du clavier
onload	Le navigateur a fini le chargement de la page

Principes de fonctionnement

- Un évènement est un objet
  - Il a des propriétés et des méthodes
- Le bouillonnement d'évènements



onchange

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    function doChange(e){
      console.dir(e);
      console.log('value change');

    }
  </script>
</head>
<body>
  <input type="text" onchange="doChange(event)">
</body>
</html>
```

onclick

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    function doClick(e){
      console.dir(e);
      console.log('button click');
    }
  </script>
</head>
<body>
  <input type="button" onclick="doClick(event)" value="click">
</body>
</html>
```

JavaScript non-intrusif

- Il ne faut pas faire référence à du JavaScript dans une page web
- WebDesigner et Développeur ne sont pas forcément les mêmes personnes !

onchange non-intrusif

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    var doLoad = function(){
      var the_input = document.getElementById('the_input');
      the_input.onchange = function(e){
        console.dir(e);
        console.log('button click');
      };
    }
    window.onload = doLoad;
  </script>
</head>
<body>
    <input type="text" id='the_input'>
</body>
</html>
```

onclick non-intrusif

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    var doLoad = function(){
      var the_button = document.getElementById('the_button');
      the_button.onclick = function(e){
        console.dir(e);
        console.log('button click');
      };
    }
    window.onload = doLoad;
  </script>
</head>
<body>
  <input id="the_button" type="button" value="click">
</body>
</html>
```

Les formulaires

- Les formulaires sont des objets :
  - **Propriétés** : action, method,…
  - **Méthodes** : submit(), reset()

Les éléments de formulaires

- Il y en a peu :
  - Input (text,file,date,…)
  - Checkbox
  - Radio
  - Select
  - Textarea
  - Button

Les éléments de formulaires

- Ils sont accessibles de différentes façons :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    window.onload = function(){
      var elm = document.forms[0].elements[2];
      var elm = document.forms.monForm.elements.monElement_3;
      var elm = document.forms.monForm.elements['monElement_3'];
    }
  </script>
</head>
<body>
  <form action="" name="monForm">
    <input type="text" name="monElement_1">
    <input type="text" name="monElement_2">
    <input type="text" name="monElement_3">
  </form>

</body>
</html>
```

Validation de formulaire

- Valider un formulaire c'est empêcher son envoi s'il n'est pas correct
- On implémente la méthode **onsubmit** qui doit renvoyer un booléen

```
<script>
  window.onload = function(){
    document.forms.monForm.onsubmit = function(e){
      var result = false;
      var elem1 = document.forms.monForm.monElement_1;

      if(elem1=='value1'){
        result = true;
      }
      else result = false;

      return result;
    }
  }
</script>
```

```
<form action="" name="monForm">
  <input type="text" name="monElement_1">
  <input type="submit" value="envoyer">
</form>
```

Le DOM

- Le DOM est une **API** de manipulation de document (balisé)
- Normalisé par le W3C
- Implémenter dans tous les langages

Récupération d'éléments

- Considérons cet extrait de page

```
<p id="p1">
  p1
  Lorem ipsum dolor sit amet, consectetur adipisicing elit.
</p>

<p id="p2" class="la_class">
  p2
  Lorem ipsum dolor sit amet, consectetur adipisicing elit.
</p>

<p id="p3" class="la_class">
  p3
  Lorem ipsum dolor sit amet, consectetur adipisicing elit.
</p>
```

Récupération d’éléments par id

```
window.onload = function(){
  var p1 = document.getElementById('p1');
}
```

Récupération d’éléments par tag

```
window.onload = function(){
  var p = document.getElementsByTagName('p');
}
```

class

```
window.onload = function(){
  var p = document.getElementsByClassName('la_class');
}
```

Récupération d'éléments par notation CSS

```
window.onload = function(){
  var p = document.querySelector('#p1');
}
```

Récupération d'éléments par notation CSS

```
window.onload = function(){
  var p = document.querySelectorAll('.la_class');
}
```

Rappels sur les notations

- **Camel case** : backgroundColor
- **Train case** : background-color

Modifier un style CSS

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    window.onload = function(){
      var p = document.getElementById('p1');
      p.style.backgroundColor = 'yellow';
    }
  </script>
</head>
<body>
  <p id="p1">
    p1
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  </p>
</body>
</html>
```

Afficher/Cacher des éléments

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script>
    window.onload = function(){
      var btn = document.getElementById('btn');
      btn.onclick = function(){
        var p1 = document.getElementById('p1');
        var visible = p1.style.visibility;
        if(visible == 'visible') p1.style.visibility = 'hidden';
        else p1.style.visibility = 'visible';
      }
    }
  </script>
</head>
<body>
  <input type="button" id="btn" value="Afficher/Cacher"><br/>
  <p id="p1">
    p1
    Lorem ipsum dolor sit amet, consectetur adipisicing elit.
  </p>
</body>
</html>
```