

# Bachelier en Informatique de Gestion

## Web : principes de base Projet de Développement Web

Enseignement supérieur économique de type court

Code FWB : 7534 29 U32 D1, 7534 30 U32 D3

Code ISFCE : 4IWPB, 4IPW3



# Table des matières

## Généralités

- 01. Introduction au web
- 03. Outils
- 05. Format XML
- 06. Format JSON

## Front-End

- 12. Structure HTML
- 13. Formulaire HTML
- 14. Mise en forme CSS
- 15. Adaptabilité
- 17. Javascript
- 18. Bibliothèque jQuery
- 19. Composant Vue.js

## Back-End

- 21. Middleware PHP
- 22. Traitement du formulaire
- 23. Architecture MVC
- 24. Données SQL
- 25. Données NoSQL
- 27. Requête asynchrone



JS



# 17. Javascript

Architecture

Variables et Typage

Structures de contrôle

Fonctions

Document Object  
Model

Évènements DOM-0

Évènements DOM-2

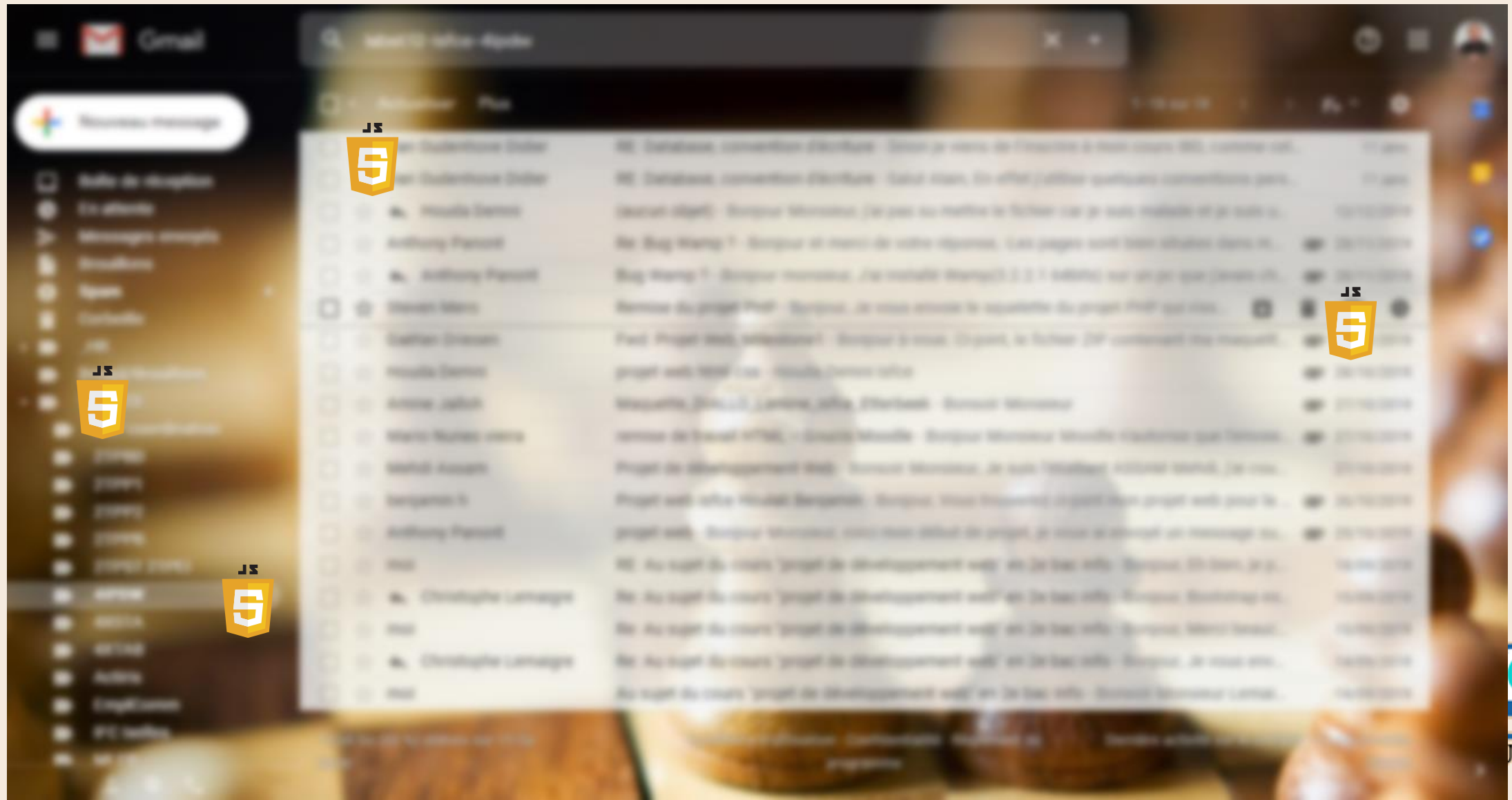


= BUROTIX 0



# Architecture

# Exemples de page web javascriptée



# Exemples d'utilisation du JavaScript

- Exemple de base:
  - <http://www-k12.atmos.washington.edu/~ovens/javascript/jseg28.html>
- Template
  - <http://www.philippagregory.com/books>
- Animation
  - <http://hereistoday.com/>
  - <http://the389.com/experiment/>
  - [http://mrdoob.com/projects/chromeexperiments/google\\_gravity/](http://mrdoob.com/projects/chromeexperiments/google_gravity/)
  - <http://gridster.net/demos/adding-widgets-dynamically.html>
- 3D
  - [http://shapejs.shapeways.com/creator/?li=devhome\\_main](http://shapejs.shapeways.com/creator/?li=devhome_main)
  - [http://mrdoob.github.com/three.js/examples/webgl\\_materials\\_cars.html](http://mrdoob.github.com/three.js/examples/webgl_materials_cars.html)

à mettre  
à jour



= BUROTIX 0

# Références

## ■ Ressources

- OpenClassRooms
- w3schools
- MDN
- Developpez.com
- CommentCaMarche.net

## ■ Outils:

- JSBin: <http://jsbin.com>
- JSFiddle: <http://jsfiddle.net/>
- Rubular: <http://rubular.com/>
- Chrome: Console debug



# Emplacement du code

- Dans une page HTML:

```
<script type="text/javascript">  
    var maVariable;  
    ...  
</script>
```

- Dans un fichier externe:

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

- Dans un attribut événement - A éviter !

```
<a onclick="var maVariable; ...">Accueil</a>
```





# Emplacement du code

- Remarque :
  - code JS exécuté par le navigateur **séquentiellement** à la lecture de la page web.
  - => Attention à l'emplacement du code.



# Emplacement du code : exo01

```
<html>
<head>
<title>Emplacement JS OK</title>
</head>
<body>
<div id="01">Bienvenue</div>
<script>
el = document.getElementById("01");
el.innerHTML = "Bonjour";
</script>
</body>
</html>
```



```
<html>
<head>
<title>Emplacement JS KO</title>
</head>
<body>
<script>
el = document.getElementById("01");
el.innerHTML = "Bonjour";
</script>
<div id="01">Bienvenue</div>
</body>
</html>
```



# Code interne/externe : exo02

- Créez une page HTML "exo02\_alert\_inline.html" contenant le script JS suivant :  
**alert("Bienvenue sur mon site");**
- Migrer ce script dans un fichier "exo02\_alert\_outline.js" et faites y appel dans votre page HTML
- Solution sur burotix.be



# Variables et Typage



= BUROTIX ()

# Typage dynamique

- JavaScript : langage à **typage dynamique**
  - Type d'une variable défini au runtime
  - Type d'une variable changeable en cours d'exécution (comme en PHP)
- Nom des variables
  - 1<sup>er</sup> caractère : une lettre, "**\_**" ou "**\$**"
  - caractères suivants : alphanumériques, "**\_**" ou "**\$**"
- Exemples:

```
var myvar = "mon texte d'initialisation";  
// ...  
myvar = 2;
```



# Mots-clés : var, let, const, ... ou rien

- `myvar = "hello";`
  - scope global
  - redéclaration possible
- `var myvar_v = "hello";`
  - scope global si déclaré globalement
  - scope local si déclaré localement
  - redéclaration possible

- `let myvar_l = "hello";`
  - scope de bloc ( `{...}` )
  - redéclaration possible (sans `let`)
- `const myvar_c = "hello";`
  - scope de bloc ( `{...}` )
  - redéclaration impossible

# Scalaires : exo03

- Boolean
  - type de valeurs possibles **true** ou **false**
- Number
  - type qui représente un **nombre**
  - forme littérale pour un nombre entier en base décimale  
**var nombreEntier = 11;**
  - forme littérale pour un nombre réel  
**var nombreReel = 11.435;**
- String
  - type qui représente une **chaîne de caractères**
  - forme littérale d'une chaîne de caractères  
**var chaineCaracteres = "des caractères";**



# Tableaux : exo04

- Forme littérale d'un tableau indexé

```
const tableau = [ "Premier élément",  
                  "Second élément" ];
```

- Forme littérale d'un tableau associatif (dictionnaire)

```
const tableauAssociatif = {  
    "cle1" : "valeur1",  
    "cle2" : "valeur2"  
};
```

- note: à l'origine de la norme JSON





# Tableaux : exo04

```
// Initialisation
var monTableau2 = ["Pierre", "Paul"];
// Ajouter un élément
monTableau2.push("Jacques");
// Lire un élément - Index commence à 0
var nom = monTableau2[2]; // Longueur du tableau
alert( "monTableau[2]=" + nom + "\n" +
      "longueur monTableau2=" + monTableau2.length );
// Tableau associatif
var monTableau3 = {};
monTableau3["Lundi"] = "Soupe";
alert("monTableau3[Lundi]=" + monTableau3["Lundi"]);
```



# Opérateurs

- Opérateurs d'affectation  
=

- Opérateurs de calcul  
+ - \* / %

- Opérateurs de comparaison  
==  
> <  
>= <=

- Opérateur de concaténation

+

- Opérateurs logiques  
! Négation  
&& ET-logique  
|| OU-logique





# Structures de contrôle

# Branchement : if, switch

- Control "**if**"

```
if( condition )
{
    ...
}
else
{
    ...
}
```

- Control "**switch case**" :

exo15-07

```
const x = 50 ;
switch(x) {
    case 1 :
        alert('1');
        break;
    case 5 :
        alert('5');
        break;
    default :
        alert('default');
        break;
}
```



# Boucle : for, while

- Boucle "**for**"

```
for ( initialisation;  
      condition;  
      incrémentation )  
{  
    ...  
}
```

- Exemple

```
for(let i=0; i<5; i++)  
{  
    alert('Itér n°' + i);  
}
```

- Boucle "**while**"

```
while( condition )  
{  
    ...  
}
```

- Exemple

```
let nombre = 0;  
while( nombre < 10 )  
{  
    nombre++;  
}
```





# Fonctions

# Fonctions

## ■ Déclaration de la fonction

Corps de  
la fonction

```
function nomFonction(arg1, arg2)  
{  
  const c = 10;  
  const result;  
  result = (arg1 + arg2) * c;  
  return result;  
};
```

Nom de la fonction

Signature de  
la fonction

Variable  
locale

## ■ Appel de la fonction

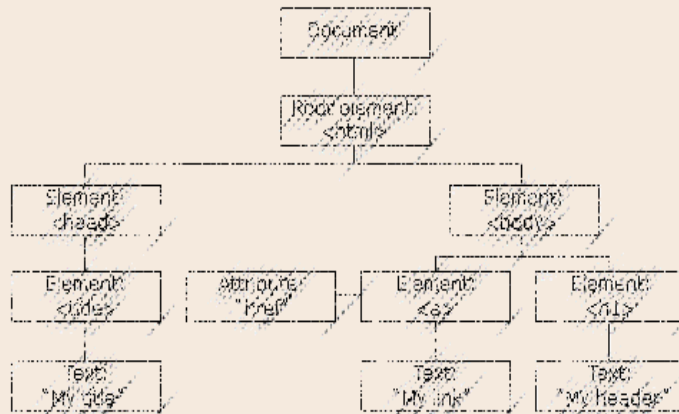
```
let a = 1;  
let b = 2;  
let res;  
res = nomFonction(a, b);
```



= BUROTIX 0



# Document Object Model



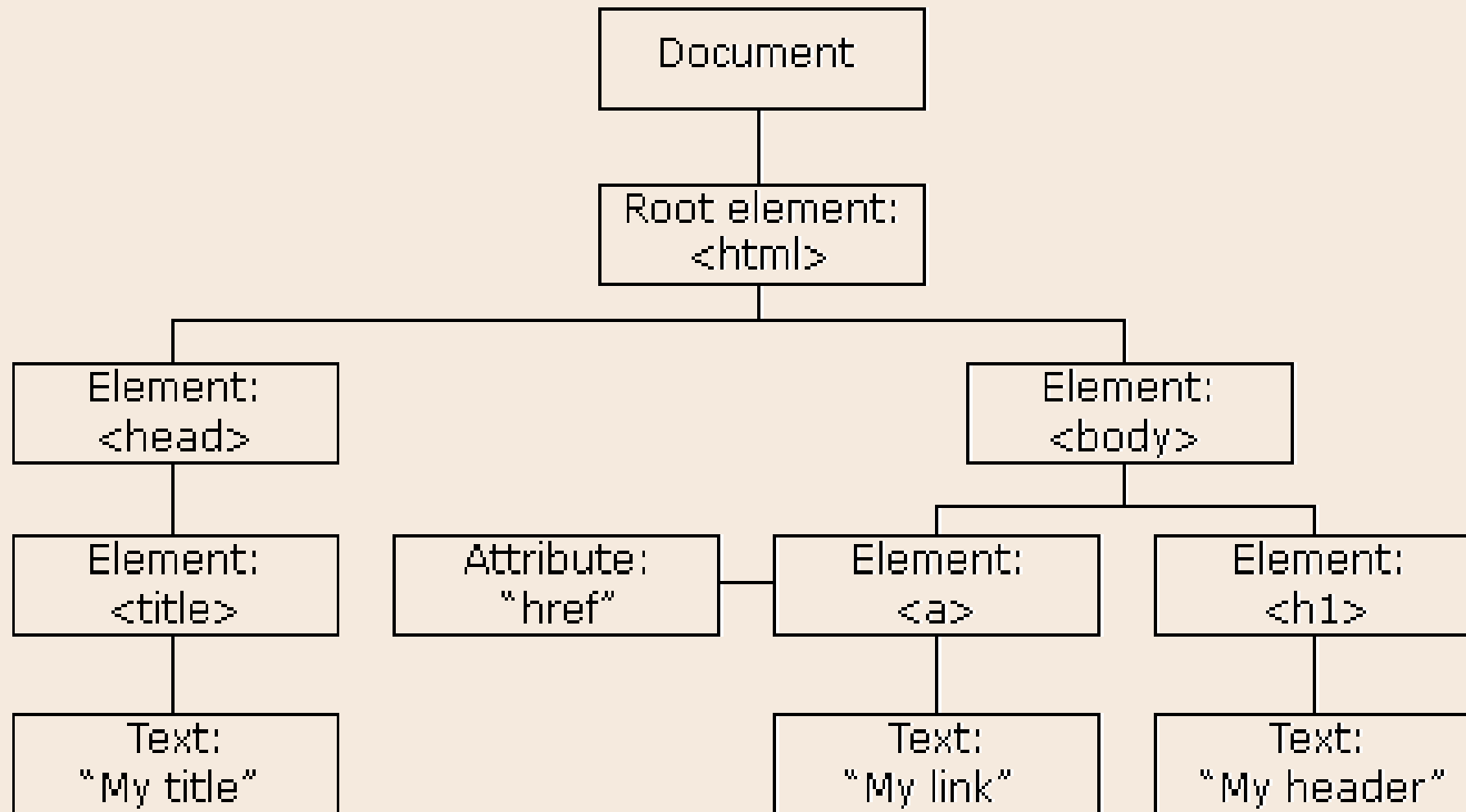


# Document Object Model

- L'API du DOM (Document Object Model) permet d'accéder à une page Web et de manipuler son contenu, sa structure et ses styles.
- DOM présente un document sous la forme d'un arbre de nœuds.
- [https://developer.mozilla.org/en-US/docs/DOM/DOM\\_Reference](https://developer.mozilla.org/en-US/docs/DOM/DOM_Reference)



# Example



# Objets du DOM

- DOM Document
  - Nœud racine du document HTML
- DOM Node
  - Arborescence du document HTML
- DOM Element
  - Contenu d'un nœud DOM
  - `p`, `div`, `a`, `table`, ...
- DOM Attribute
  - Attribut d'un élément HTML
  - `href`, `id`, `class`, ...



# DOM Document

- L'objet Document est l'élément racine d'un document (ex. page web, document XML)
- Il hérite des méthodes et propriétés de l'objet Noeud (cf. slides suivants)



# DOM Document - Méthode

- `getElementById(elementID)`
  - Retourner l'élément ayant l'attribut id spécifié
  - Param : `elementID` : string
  - Return: un élément
- Exemple:  

```
let o = document.getElementById("demo");
```



# DOM Document - Méthode

- **getElementsByTagName(tagName)**
  - Retourner une collection d'éléments ayant le nom spécifié
  - Param : **tagName**: string
  - Return : liste d'éléments
- Exemple:

```
var nl = document.getElementsByTagName("div");
for( var cle in nl )
{
    alert( cle + ": " + nl[cle].id );
}
```



# DOM Document - Méthode

- `querySelector(cssSelector)`
  - Retourner l'élément répondant au sélecteur spécifié
  - Param : `cssSelector` : string
  - Return : un élément
- Exemple :  
`document.querySelector(".exampleclass");`



# DOM Document - Méthode

- `querySelectorAll(cssSelector)`
  - Retourne une collection d'éléments répondant au sélecteur CSS
  - Param : `cssSelector`: string
  - Return : liste d'éléments
- Exemple:  
`document.querySelectorAll("div.red h1");`



# DOM Document - Méthode

- `createElement(nodeName)`

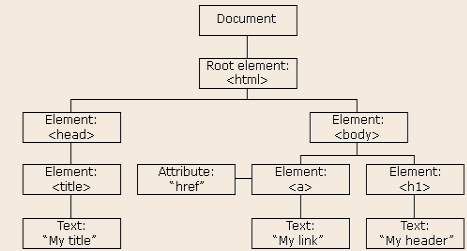
- Créer un élément et le retourner
- Param : `nodeName` : string
- Return : un élément

- Exemple:

```
var btn = document.createElement("h1");
```

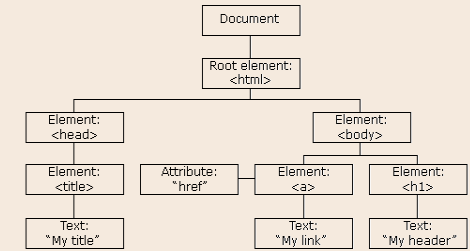


# DOM Node



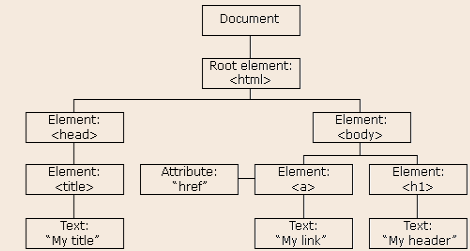
- Représenter un nœud dans un document HTML
- Il existe 12 types de nœuds HTML, dont:
  - Element
  - Attr
  - Text
  - Comment
- <https://developer.mozilla.org/en-US/docs/Web/API/Node.nodeType>

# DOM Node - Propriétés



- **textContent**
  - Retourner le texte d'un nœud et de ses descendants

# DOM Node - Méthode



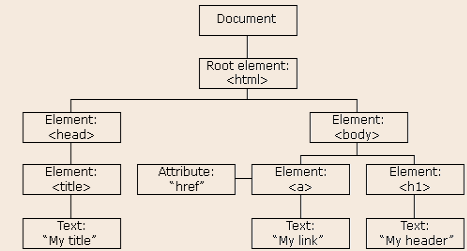
- `appendChild(node)`

- Ajouter un nœud en tant que dernier enfant
- Param : **node** : un nœud
- Return : un nœud

- Exemple 1 :

```
var d = document.createElement("div");  
d.innerHTML = "je suis un nouvel élément";  
var b = document.getElementsByTagName("body")[0];  
b.appendChild(d);
```

# DOM Node - Méthode



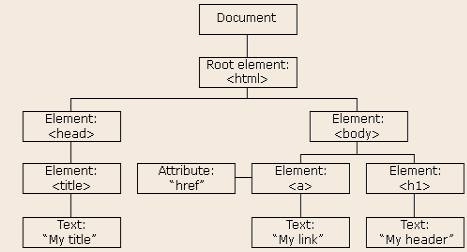
- Que fait le code JS suivant ? Exemple 2

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>
<script>
  var para = document.createElement("p");
  para.innerHTML = "This is not new.";
  var element = document.getElementById("div1");
  element.appendChild(para);
</script>
```

- [https://www.w3schools.com/js/js\\_html\\_dom\\_nodes.asp](https://www.w3schools.com/js/js_html_dom_nodes.asp)



# DOM Node - Méthode

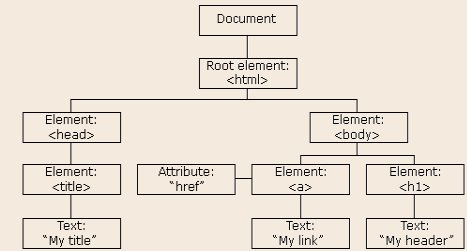


- **insertBefore(newNode, existingNode)**
  - Ajouter un nœud juste avant le nœud enfant spécifié
  - Param : **newNode**: un noeud
  - Param : **existingNode**: un noeud
  - Return : un noeud

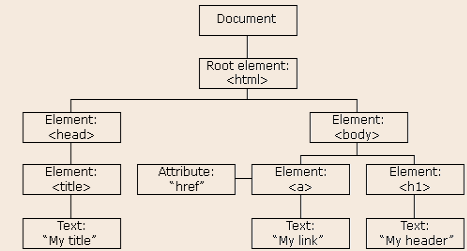
# DOM Node - Méthode

- Que fait le code JS suivant ? Exemple 3

```
<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
<script>  
  var para = document.createElement("p");  
  para.innerHTML = "This is not new.";  
  var element = document.getElementById("div1");  
  element.appendChild(para);  
  var child = document.getElementById("p1");  
  element.insertBefore(para, child);  
</script>
```



# DOM Node - Méthode



- `removeChild(node)`
  - Supprimer le nœud enfant spécifié
  - Param : **node** : un nœud
  - Return : un nœud

- Exemple :

```
var o=document.getElementById("myList");  
o.removeChild(o.childNodes[0]);
```



= BUROTIX 0



# DOM Element

- Manipuler les attributs

Méthode	Paramètre	Description
getAttribute	Identifiant de l'attribut	Référence un attribut d'un élément en utilisant son identifiant.
hasAttribute	Identifiant de l'attribut	Détermine si un attribut est présent pour un élément.
removeAttribute	Identifiant de l'attribut	Supprime un attribut pour un élément.
setAttribute	Identifiant de l'attribut ainsi que sa valeur	Crée un attribut ou remplace un attribut existant d'un élément.



# DOM Element - Propriétés

- `<n>.setAttribute(attrName, attrValue)`
  - Ajouter l'attribut `attrName` avec la valeur `attrValue` au nœud `<n>`.
  - Param : `attrName` : le nom de l'attribut
  - Param : `attrValue` : la valeur de l'attribut
- Exemple : Pour appliquer la classe "democlass" au premier élément H1 du document

```
document.getElementsByTagName("H1")[0].
  setAttribute("class", "democlass");
```



# DOM Element - Propriétés

- **previousElementSibling**
  - Retourner le nœud élément "frère" précédent (du niveau identique au nœud courant)
- **nextElementSibling**
  - Retourner le nœud élément "frère" suivant (du niveau identique au nœud courant)



# DOM Element - Propriétés

- Que fait le code JS suivant ? Exemple 4

```
<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>  
<script>  
  var e = document.getElementById("p1");  
  var f = e.nextElementSibling;  
  f.innerHTML = "This is a new text";  
  var g = f.previousElementSibling;  
  g.innerHTML = "This is not a new text";  
</script>
```



# DOM Element - Propriétés

- **childElementCount**
  - Retourner le nombre d'éléments enfants
- **firstElementChild**
  - Retourner le premier élément enfant d'un noeud
- **lastElementChild**
  - Retourner le dernier élément enfant d'un noeud
- **children**
  - Retourner une collection (HTMLCollection) contenant les éléments enfants d'un élément



# DOM Element - Propriétés

- Que fait le code JS suivant ? Exemple 5

```
<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
  <p id="p3">This is a third paragraph.</p>  
</div>  
<script>  
  var e = document.getElementById("div1");  
  var n = e.childElementCount;  
  e.children[n-2].innerHTML = "this is new";  
</script>
```



# DOM HTML Element

- **innerHTML**

- Accéder ou remplacer complètement le contenu d'un élément par celui spécifié dans une chaîne de caractères.





# DOM NodeList - Propriété



- **length**
  - Retourner le nombre de noeud dans une collection





# Application : tableur : exercice 21

- Fichier : exo21\_start.php
- Développez un tableur en javascript  ... qui doit seulement faire l'addition de trois cellules. 
- Le total doit être remis à jour automatiquement si une des trois cellules est modifiée.

B4					  =	=SOMME(B1:B3)
	A	B	C	D		
1	Valeur 1	34				
2	Valeur 2	65				
3	Valeur 3	23				
4	<b>TOTAL</b>	<b>122</b>				
5						

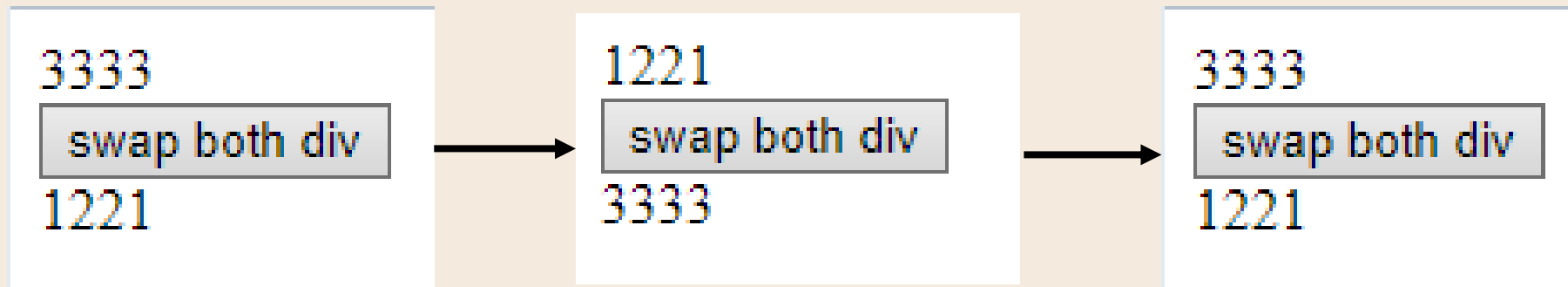
# Application : le "div filler" : exo 22

- Fichier : exo22\_start.html
- Affichez dans une **alert** la liste des **id** des **div**.
- Créez par Javascript
  - un nouveau **div** à la fin du document
  - **id** : "nouveau"
  - **innerHTML** : "je suis un nouvel élément"
- Faites que le bouton "**ajouter**" ajoute le contenu de l'input "**ifContenu**" au div "**nouveau**".



# Sibling : exo 24

- Fichier : exo24\_start.html
- Voici en image ce qui doit se passer quand on clique sur le bouton : inversion du contenu des deux **div**.





Évènement

# Evènement : le focus

- Propriété d'un élément ciblé
- Un élément ciblé reçoit tous les événements de votre clavier.
  - Seulement `<input>`, `<select>`, `<textarea>` et `<a>` (et `<button>`)
- Exemple : `<input>`
  - si vous cliquez dessus alors l'input possède le focus
  - si vous tapez des caractères sur votre clavier, alors vous les voyez s'afficher dans l'input en question.



# Evènement : les types

Nom	Attribut	Action déclencheuse
<b>Click</b>	<b>onclick</b>	Click de souris sur un élément
Dblclick	<b>ondblclick</b>	Double click sur un élément
<b>Focus</b>	<b>onfocus</b>	L'élément reçoit le focus
<b>Blur</b>	<b>onblur</b>	L'élément perd le focus
<b>Change</b>	<b>onchange</b>	Le contenu d'un champ est modifié (élément SELECT RADIO)
Input		Taper un caractère dans un champ de texte
<b>Select</b>	<b>onselect</b>	Du texte est sélectionné (élément INPUT, TEXTAREA)
Keydown	<b>onkeydown</b>	Une touche est pressée
KeyPress	<b>onkeypress</b>	Une touche de caractère est pressée
Keyup	<b>onkeyup</b>	Une touche est relâchée

Nom	Attribut	Action déclencheuse
<b>Load</b>	<b>onload</b>	La page ou l'image est chargée
Unload	<b>onunload</b>	L'utilisateur sort de la page
Resize	<b>onresize</b>	La taille de la fenêtre est réajustée
Mousedown	<b>onmousedown</b>	Le bouton de la souris est pressé
Mousemove	<b>onmousemove</b>	La souris est bougée
Mouseout	<b>onmouseout</b>	La souris sort d'un élément
<b>Mouseover</b>	<b>onmouseover</b>	La souris survole un élément
Mouseup	<b>onmouseup</b>	Le bouton de la souris est relâché
Error	<b>onerror</b>	Si une erreur apparaît lors du chargement de la page, d'une image...

# Evènement & HTML

- Méthode historique mais pédagogique
  - Attribut spécifique à placer dans chaque élément soumis à un évènement
- Exemple

```
<button onclick="maFunction()">
...
</button>
```
- Exemple avec `this`

```
<button onclick="alert('Voici le contenu de l\'élément
que vous avez cliqué :\n\n' + this.innerHTML);">
    Cliquez-moi !
</button>
```



# Evènement : exo 31

- Fichier : exo31\_start.php
- Start : l'utilisateur peut appuyer sur "validate" même avec l'input non rempli.
- Solution : vérifier que l'utilisateur a bien renseigné une adresse mail dès qu'il clique sur le bouton "validate".
- Tuyau : **onblur**



# Conflit entre événements : exemple 33

- Déterminez dans le code ci-dessous si l'utilisateur atterrira sur le site indiqué ou non.
- C-à-d : A votre avis, cliquer sur le lien revient-il
  - À suivre le lien ?
  - À exécuter le code JS ?
  - À faire les deux ? Mais alors, dans quel ordre ?
- Déduisez-en le sens de `return`.

```
<a href="http://www.burotix.be"  
  onclick="alert('Clic !'); return false;">  
  Lien vers burotix.be  
</a>
```





# Évènement & DOM-0



= BUROTIX ()

# DOM-0 par l'exemple

```
HTML { <button id="clickme">
      Cliquez-moi !
      </button>
      .....
JS { <script>
    var element = document.getElementById('clickme');
    element.onclick = function() {
        alert("Vous m'avez cliqué !");
    };
    </script>
```

propriété

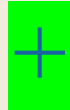
- On récupère tout d'abord l'élément HTML dont l'ID est "clickme";
- On accède ensuite à sa propriété `onclick` à laquelle on assigne une fonction anonyme ;

# DOM-0 par le principe

- On définit les événements non plus dans le code HTML mais directement en JavaScript.
- Un événement de chaque événement standard se traduit par une propriété dudit élément dont le nom est précédé par les deux lettres « on ».
- Cette propriété prend pour valeur
  - soit le nom d'une fonction
  - soit une fonction anonyme avec un code fourni immédiatement



# DOM-0 : évaluation



- Pratique et simple



- Vieux (sic)
- Impossible de créer plusieurs fois le même événement






Evènement & DOM-2



# DOM-2 par l'exemple

HTML { `<button id="clickme">`  
          `Cliquez-moi !`  
          `</button>`  
          .....  
JS { `<script>`  
      `var element = document.getElementById('clickme');`  
      `element.addEventListener("click", function() {`  
          `alert("Vous m'avez cliqué !");`  
      `});`  
      `</script>`



méthode

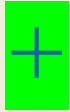
- On récupère l'élément HTML dont l'ID est "**clickme**";
- On utilise la méthode **addEventListener()** avec comme paramètres
  - nom de l'événement (sans les lettres « on ») ; p.ex.: "**click**"
  - fonction à exécuter (nommée ou anonyme);

# DOM-2 par l'exemple, une alternative

```
HTML { <button id="clickme">
        Cliquez-moi !
    </button>
    .....
JS { <script>
    var element = document.getElementById('clickme');
    var monEvenement = function() {
        alert("Vous m'avez cliqué !");
    };
    element.addEventListener( "click" , monEvenement );
    </script>
```



# DOM-2 : évaluation



- Création multiple d'un même évènement
- Gestion de l'objet **event**.
- A utiliser surtout lors de l'intégration de librairies multiples au sein de votre site web !



- **Lourdeur du code**



# Évènements multiples

```
<button id="clickme">
    Cliquez-moi !
</button>

.....
<script>
var element = document.getElementById('clickme');
// Premier événement click
element.addEventListener('click', function() {
    alert("Vous m'avez cliqué ! Et de un ! ");
});
// Deuxième événement click
element.addEventListener('click', function() {
    alert("Vous m'avez cliqué ! Et de deux ! ");
});
</script>
```

- Ordre de déclenchement aléatoire, fonction du navigateur.
  - Peut-être dans l'ordre de création (mais pas sûr)



# Suppression d'un évènement

- Méthode `removeEventListener()`

```
// On crée l'évènement  
element.addEventListener('click', myFunction);  
// On supprime l'évènement  
// en lui repassant les mêmes paramètres  
element.removeEventListener('click', myFunction);
```



# Propagation d'un évènement : *capture or bubbling?*

`<div>`

`<button> Du texte ! </button>`

`</div>`

- Si on attribue
  - un évènement `click` à `<div>`, et
  - un évènement `click` à `<button>`,
- Si on clique sur "`Du texte !`" ...
- Quel évènement se déclenchera-t-il en premier ?



# Propagation d'un évènement : *capture or bubbling*? Réponse.

```
<div>
```

```
    <button>Du texte !</button>
```

```
</div>
```

- Mode *capture* :
  - d'abord l'évènement du `<div>` est activé
  - ensuite celui du `<button>`
- Mode *bubbling* :
  - d'abord l'évènement du `<button>` est activé
  - ensuite celui du `<div>`
- Par défaut : *bubbling*



# Propagation d'un évènement : exo40

```
<div id="capt1">
  <span id="capt2">capture</span>
</div>
<div id="boul1">
  <span id="boul2">bouillonnement</span>
</div>

<script>

var capt1 = document.getElementById('capt1'),
    capt2 = document.getElementById('capt2'),
    boul1 = document.getElementById('boul1'),
    boul2 = document.getElementById('boul2');

capt1.addEventListener('click', function() {
  alert("événement div déclenché.");
}, true);
```

```
capt2.addEventListener('click', function() {
  alert("événement span déclenché.");
}, true);
```

```
boul1.addEventListener('click', function() {
  alert("événement div déclenché.");
}, false);
```

```
boul2.addEventListener('click', function() {
  alert("événement span déclenché.");
}, false);
```

- La méthode **addEventListener** comporte un troisième paramètre, de type boolean :
  - **True** si mode capture
  - **False** si mode bubbling (par défaut)



Objet "Event"



= BUROTIX ()

# L'objet "Event"

- Utilité : Fournir les informations sur l'événement déclenché, par ex. :
  - touches enfoncées
  - coordonnées du curseur
  - élément qui a déclenché l'événement, ...
- Accessible seulement
  - lorsqu'un événement est déclenché
  - via une fonction exécutée par l'événement concerné



# L'objet "Event"

- Exemple:

```
element.addEventListener('click', function(e) {  
    // Affiche le type de l'événement (click, etc.)  
    alert(e.type);  
});
```

- Argument « **e** » : référence vers l'objet « **Event** ».



# L'objet "Event" : propriétés

- **type**
  - type de l'événement (click, mouseover, etc.)
- **target**
  - élément déclencheur de l'événement
- **clientX, clientY**
  - position du curseur
- **keyup, keydown**
  - touche quelconque frappée
- **keypress**
  - touche textuelle frappée

