# Amélioration progressive

felipe@ift3225 Hiver 2020

Grandement (entièrement) inspiré de


CSS

# Principe

```
p {
  color: red;
  font-weight: bold;
}
```

- Sources de problèmes dans un tel bloc
  1. Propriété non reconnue (ex: `colour`)
  2. Problème dans le sélecteur
  3. Une valeur peut ne pas être reconnue (ex: `rgba(180,246,248,.43)`)

**Si une erreur survient dans une règle, la règle est ignorée, si une erreur survient dans le sélecteur, tout le bloc est ignoré**

# Mise en pratique

```
p {
  background-color: rgb(137,224,160);
  background-color: rgba(180,246,248,.43);
}
```

Certains navigateurs ne reconnaissent pas la seconde valeur de propriété

- La première est une règle par défaut
- Si l’agent reconnaît rgba, alors le style est remplacé

→ **L’ordre des règles est important**

# Zen Garden

```
#intro {
  margin: 0;
  padding: 0;
  width: 660px;
  height: 80px;
  background: transparent url(introkop.gif)
  no-repeat right top;
}
```

pour les « vieux » navigateurs

Le sélecteur [ ] n'est pas reconnus d'IE6 qui ignorera donc le second bloc dont le CSS est plus pointu

```
/* … */
```

pour les navigateurs plus modernes

```
body[id=css-zen-garden] #intro {
  position: absolute;
  top: 0;
  left: 0;
  float: none;
  margin: 0;
  width: 100%;
  height: 350px;
  background: none;
}
```

# Ou mieux

```
#intro {
  /* Old Layout */
}

/* … */

[foo], #intro {
  /* Advanced Layout */
}
```

[foo] sert juste à vérifier si le navigateur comprend le sélecteur (foo n’est pas un attribut dans la page de Zen Garden) . La sélectivité du sélecteur dans le second bloc est le même que celle du premier (la cascade s’applique).

# Note

```
p,
p:not([title]) {
  color: red;
  font-style: bold;
}
```

**Tous les navigateurs n’implémentent pas le sélecteur not**

```
p {
  color: red;
  font-style: bold;
}

p:not([title]) {
  // idem or more specific
}
```

# De l’ordre !

On veut que certaines images (celle(s) marquée(s) `focal`) apparaissent avec un cadre plus épais

```
<figure class="frame focal">
    <img class="inner" src="i/lodge.jpg" alt="" title="…"/>
</figure>
 <figure class="frame">
     <img class="inner" src="i/room.jpg" alt="" title="…"/>
</figure>
```

```
.frame {
  margin: 0 auto 40px;
}

.focal {
  margin: 0 20px 25px 30px;
}
```

**Ces deux sélecteurs ont la même spécificité => l’ordre est important**

# De l’ordre dans vos styles

```
/* =Typography */
img {
  display:block;
}

/* =Layout */
@media screen {
  .frame .inner {
    border: 10px solid;
  }
}

/* =Color */
.frame .inner {
  background-color: rgb(227, 222, 215);
  border-color: rgb(227, 222, 215);
}
```

**Vous devriez styler en respectant cet ordre.**

## Pourquoi?

- `border` est un raccourci pour `border-top`, `border-right`, etc.
- chacune de ces propriétés est également un raccourci, par ex. `border-top: 10px: solid :`
  - `border-top-width: 10px`
  - `border-top-style: solid`
  - `border-top-color: inherit`

**=> les propriétés de mise en page modifient les couleurs**

# De l’ordre dans vos styles

```
/* =Color */
a:link, a:visited {
  color: rgb(180, 49, 25);
}

a:hover {
  color: rgb(235, 123, 35);
}
/* … */

/* =Effects */
@media screen {
  a {
    transition-duration: .5s;
  }
}
```

# De l’ordre partout

```
<link rel="stylesheet" href="main.css"/>
<!--[if IE 9]><link rel="stylesheet" href="ie9.css"/><![endif]-->
<!--[if lte IE 8]><link rel="stylesheet" href="ie8.css"/><![endif]-->
<!--[if lte IE 7]><link rel="stylesheet" href="ie7.css"/><![endif]-->
<!--[if lte IE 6]><link rel="stylesheet" href="ie6.css"/><![endif]-->
```

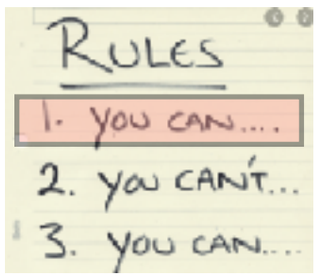
- IE9 est pas mal plus conformant à la norme que les versions plus vieilles
- Les règles pour IE8 pourraient être utiles pour IE7 (et en cas de conflit, celles de IE7 auront préséance)
- …


JavaScript

# Idée

Le contenu de votre page devrait être servi **même en l’absence de javascript ou de certaines librairies** (JQuery, Bootstrap, etc.) sur lesquelles elle s’appuie.

À l’aide de quelques **règles** relevant du bon sens, et de la **bonne pratique**




# Tout le contenu doit rester accessible sans javascript

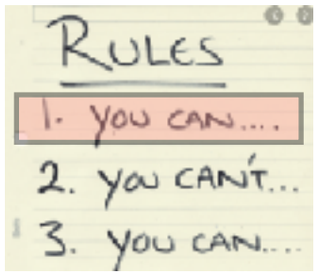
- Ex: (`newWin` n'est pas une fonction standard)

```
<a href="#" onclick="newWin( 'http://easy-designs.net/' );">Easy! Designs</a>
```

technique connue sous le nom de Hijax

**Problème:** requiert javascript car la propriété `href` n'est pas renseignée
**Solution:** donner une valeur à `href` !

```
<a href="http://easy-designs.net/"
    onclick="newWin( this.href ); return false;">Easy! Designs</a>
```



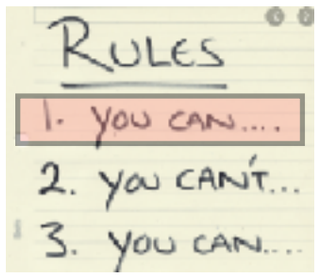

# Ok, mais…

on doit faire cela pour chaque lien que l'on veut détourner.

**Solution:** marquer l'ancre avec la propriété `rel` souvent utilisée pour qualifier le lien sémantique avec l'URL indiquée (`class` ferait aussi l'affaire, bien que moins précis)

```
<a rel="external" href="http://easy-designs.net/">Easy! Designs</a>


var links = document.getElementsByTagName( 'a' ),
      rel, i = links.length;

while ( i-- ) {
  rel = links[i].getAttribute( 'rel' );
  if ( rel && rel.match( /\bexternal\b/ ) ) {
    links[i].onclick = function() {
      newWin( this.href );
      return false;
    };
  } // if
} // while
```

**Enregistrement d'un gestionnaire pour chaque lien marqué `external` par la propriété `rel`**




# Presque parfait, mais…

- Que se passe t-il si on crée à la volée (par exemple via AJAX) de nouveaux liens ?
  - La boucle  précédente a déjà été appelée
- On peut bien sûr  ajouter un gestionnaire pour ces nouveaux liens
  - Pénible …
- Ou mieux (pro) utiliser la délégation d'événements

# Gestion d’un événement

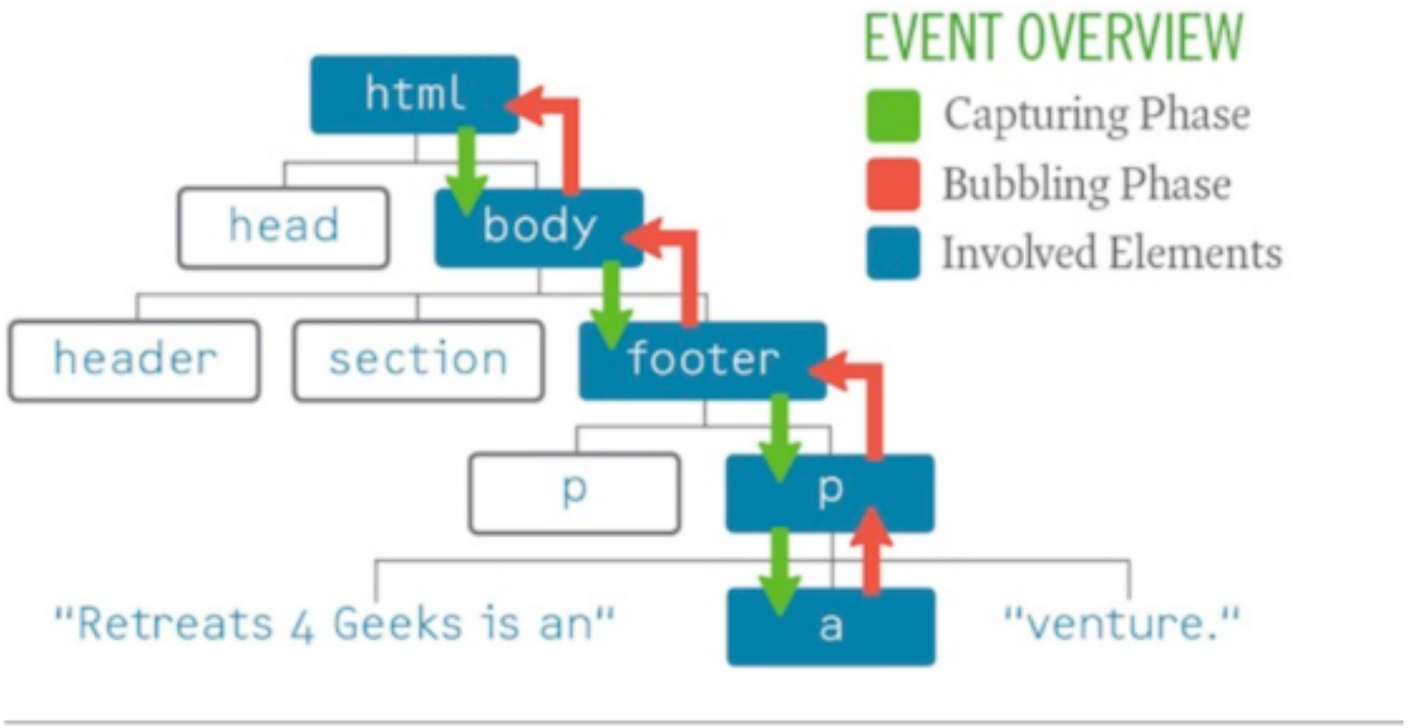



Figure 4.2: *Event capturing and bubbling.*

- 2 phases:
  - capture
  - bullage

**addEventListener**(`eventName`, `function`, `mode`)

`mode`: false (défaut = enregistrement lors du bullage)
true (enregistrement lors de la capture)

Note: L’événement passé dans les différents noeuds encode l’information pertinente (l’élément concerné).

# Délégation

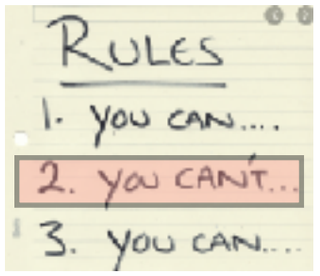
On délègue la gestion de l'événement à un parent qui abrite les éléments concernés (ici les liens) et potentiellement ceux ajoutés à la volée. Au pire, on peut déléguer à l'élément `<body>`

```
document.body.onclick = function( e ) {

  // homogénéisation (IE versus W3C)
  e = ( e ) ? e : event;
  var el = e.target || e.srcElement, rel;

  // external links
  rel = el.getAttribute( 'rel' );
  if ( el.nodeName.toLowerCase() == 'a' &&
       rel && rel.match( /\bexternal\b/ ) ) {

    newWin( el.href );

    // cancel the default action (IE versus others)
    if (e.preventDefault)
      e.preventDefault();
    else
      e.returnValue = false;
  } // if
};
```

Élément recevant l'événement

Hijax

Éviter la propagation de l'événement




# Créez les éléments nécessaires à la logique de javascript en javascript

**Étude de cas**


**<nav>**

Parfait lorsqu'il y a de la place sur le *viewport*

# Gist de l’implémentation

Code jQuery (que nous aborderons) prochainement

```
var
// reference the window
$window = $(window),
// get the navigation
$old_nav = $('#top nav > *'),
// get the links
$links = $old_nav.find('a'),
// track what’s showing
showing = 'old',
// the browser width that triggers the change
trigger = 765,
// we’ll use these shortly
$new_nav, $option,
// we’ll need a timer too
timer = null;
```

Notez les déclarations avec un seul `var`

## Mémorisation des sélecteurs

- $(*selector*) vous donne accès en jQuery au language des sélecteurs CSS (la vie est belle).

## Variables diverses

- convention: $*id* indique une variable dépendante de jQuery

# Gist de l'implémentation

Code jQuery (que nous aborderons) prochainement

```
$new_nav = $('<select></select>');
$option = $('<option>-- Navigation --</option>')
          .appendTo($new_nav);
```

Création d'un élément <select>

- Tellement plus agréable que `createElement`

```
$links.each(function(){
  var $a = $(this);
  $option.clone()
    .attr( 'value', $a.attr('href') )
    .text( $a.text() )
    .appendTo( $new_nav );
});
```

Recopie dans des élts <option> des liens de la barre de nav.

- La syntaxe jQuery peut intimider, mais rien que quelques heures de pratique ne règleront pas

**Exercice:** faire le code DOM équivalent

# Gist de l'implémentation

Code jQuery (que nous aborderons) prochainement

```
$new_nav = $new_nav
        .wrap('<div id="mobile-nav"/>')
        .parent()
        .delegate( 'select', 'change',
        function(){
           window.location = $(this).val();
        }
);
```

$new_nav

A online everything

- La philosophie jQuery

Output:

```
<div id= « mobile-nav">
  <select>
     <option>-- Navigation —</option>
     <option>Events</option>
     <option>About</option>
     <option>Contact</option>
  </select>
</div>
```

**Exercice:** faire le code DOM équivalent

# Gist de l’implémentation

Code jQuery (que nous aborderons) prochainement

```
function toggleDisplay() {

  var width = $window.width();

  if ( showing == 'old' &&
       width <= trigger ) {

    $old_nav.replaceWith($new_nav);
    showing = ‘new';

  }
  else if ( showing == 'new' &&
            width > trigger ) {

    $new_nav.replaceWith($old_nav);
    showing = ‘old';

  }

}
```

*Swap* de la DOM selon le *viewport*

**Yeh …**

- Plus agréable que de passer par les fonctions de la DOM comme `replaceChild()`

# Gist de l’implémentation

Code jQuery (que nous aborderons) prochainement

```
toggleDisplay(); // initialize the right view

$window.resize(
  function(){
    if ( timer ) clearTimeout(timer);
    timer = setTimeout( toggleDisplay, 100 );
  }
);
```

Installation du gestionnaire

- Quand on resize une fenêtre il y a typiquement des centaines d’événements générés (le temps que l’on garde la souris enfoncée)

**Création d’un timer qui retarde l’appel de 100ms. Si la fenêtre est encore en cours de modification (souris non relevée que lorsque l’événement est lancé, le timer est détruit (pas d’appel à `toggleDisplay`) et un nouveau timer est crée (pour à nouveau 100ms d’attente).**

Évite simplement plein d’appels à `toggleDisplay` qui force le rafraîchissement du viewport

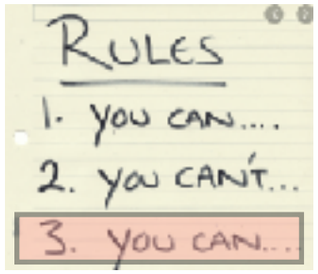
# Ne présumer de rien !

## Et que se passe t-il sinon ?

- Soit du code résilient en plus à écrire (ex: pas de jQuery, alors code avec la bonne vieille DOM)

- Au pire, l'élément  <nav> sera (si HTML5) affiché et l'utilisateur pourra toujours se débrouiller.

```
if ( typeof( jQuery ) != 'undefined' )
{
  /* Existing code goes here */
}

if ( document.getElementById )
{
  /* Code using document.getElementById() goes here */
}

if ( $old_nav.length && $links.length )
{
  /* We know the DOM elements we need exist and can do something with them */
}
```




# Appliquer les styles au moment opportun

**Étude de cas**


**<select>**

Plus compact sur petit *viewport*

On pourrait encoder dans le HTML deux structures: <nav> et <select> puis cacher l'une d'elle et *swapper* au besoin.

**Cons:**

- deux fois la même structure (moins maintenable)
- HTML contaminé  (moins SEOisable)

=> Encoder au besoin le marquage HTML dans javascript

**Note:** pris en charge par exemple dans bootstrap (*yet another* dépendance…)