1 CSS (4 points)

Soit une page web composée de trois blocs de texte cachés par des carrés gris foncé. Le code HTML, le fichier de style et l'affichage désiré sont illustrés aux figures **??**a, **??**b et **??**c.

a) (1 point) Vous remarquez que dans la deuxième règle, nous avons utilisé le sélecteur `".masque > .boite"`. Nous aurions aussi pu utiliser le sélecteur `".masque .boite"`. Expliquez pourquoi notre choix est préférable.

Cela rend plus efficace la recherche, puisqu'il nest pas nécessaire de considérer tous les chemins possible de descendance entre les deux items.

b) (1 point) On voudrait que, lorsqu'on clique sur un carré, celui-ci se rabatte vers le bas (apparaissant alors en gris clair), dévoilant ainsi le texte caché, tel qu'illustré à la figure **??**d. Ajoutez la règle CSS nécessaire pour obtenir ce comportement.

```
.boite:active {
  background-color: #ccc;
  position: relative;
  top: 50px;
}
```

c) (1 point) Sans rien changer au source HTML, quel serait la règle CSS qu'il faudrait ajouter pour que les carrés gris n'apparaissent pas, dévoilant ainsi les contenus.

```
.masque .boite {
  display: none;
}
```

d) (1 point) Expliquez la raison-d'être de la dernière règle, qui fixe une valeur négative à la propriété *z-index*.

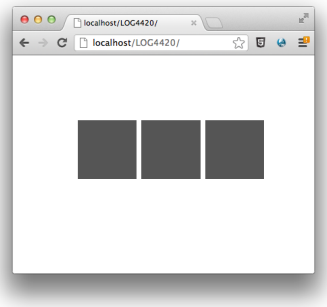
Pour que le contenu ne soit pas superposé au carré gris, on le déplaceen arrière-plan. Comme la background est transparent par défaut, il sera visible lorsque le carré gris sera déplacé.

```html
<!doctype html>
<html>
  <head>
    <meta charset="utf8">
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="conteneur">
      <div class="masque">
        <div class="boite"></div>
        <div class="boite"></div>
        <div class="boite"></div>
      </div>
      <div class="cache">
        <div class="boite"> Un</div>
        <div class="boite"> Deux</div>
        <div class="boite"> Trois</div>
      </div>
    </div>
  </body>
</html>
```

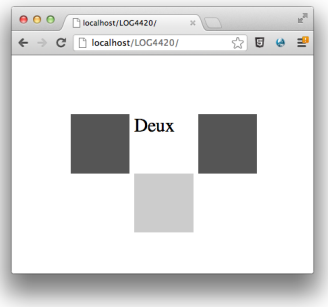
(a) Source HTML

```css
.boite {
    width: 50px;
    height: 50px;
    display: inline-block;
}
.masque > .boite {
    background-color: #555;
}
.conteneur {
    position: absolute;
    width: 158px;
    height: 100px;
    left: 50px;
    top: 50px;
}
.cache, .masque {
    position: absolute;
    top: 0;
    left: 0;
}
.cache { z-index: -10; }
```

(b) Ébauche du fichier *style.css*



(c) Affichage initial



(d) Lorsqu'on clique sur la boîte du milieu

FIGURE 1 – Code source et résultat désiré pour la question **??**b.

2 DOM et Javascript (4 points)

a) (1 point) Dites quelle est la différence entre l'objet `this` de javascript et l'objet `$(this)` de jQuery et indiquez la principale conséquence de cette différence ?

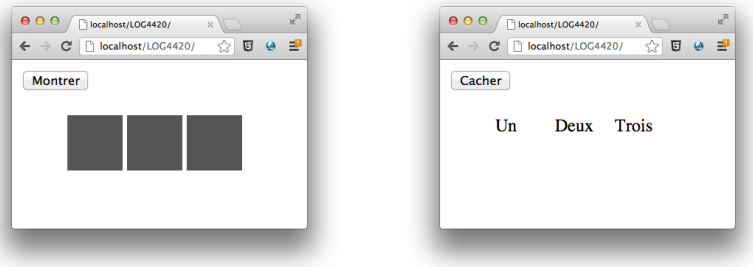
L'objet `$(this)` est un nouvel objet créé par jQuery à partir de `this` qui se comportent comme une collection (ne contenant qu'un seul item dans ce cas). La principale conséquence est que les opérations standard du DOM ne pourront être appelée directement à partir de cet objet.

b) (3 points) Reprenons l'exemple de la question précédente. On voudrait maintenant ajouter un bouton qui permet de révéler tout ce qui se cache derrière les carrés noirs. Plus précisément, lorsqu'on clique sur le bouton "Montrer" (figure **??**a), les carrés noirs disparaissent. Le bouton devient alors un bouton "Cacher". Si on clique à nouveau sur ce bouton (figure **??**b), les carrés noirs réapparaissent et le bouton redevient le bouton "Montrer" original.

Pout obtenir ce comportement nous ajouterons un seul élément dans le source HTML, directement sous la balise *body* :

```
<button id="toggle">Montrer</button>
```

Complétez le travail en modifiant le CSS et en ajoutant du code javascript ou jQuery. Vous ne pouvez apporter aucune autre modification au code HTML.



(a) Affichage initial (b) Après avoir cliqué sur le bouton

FIGURE 2 – Résultat désiré pour la question **??**.

Dans le CSS, il faut ajouter une classe :

```
.masque .boite.montrer { display: none; }
```

Finalement, il faut écrire le script pour l'interaction avec le bouton :

```
function montrer() {
  $('.masque .boite').addClass('montrer')
  $('button').text('Cacher').click(cacher)
}

function cacher() {
```

```
    $('.masque .boite').removeClass('montrer')
    $('button').text('Montrer').click(montrer)
  }

  $(document).ready(function() {$('#Montrer').click(montrer)})
```

3 Ajax (2 points)

Soit la vue suivante en Lift :

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8" >
  </head>
  <body class="lift:content_id=real_content">
    <div class="lift:surround?with=default;at=content" id="real_content">
      <div class="conteneur"  data-lift="ReponseAjax">
        <button id="Changer"></button>
        <div class="contenu"><p>Message</p></div>
      </div>
    </div>
  </body>
</html>
```

La classe ReponseAjax est définie de la manière suivante :

```
class ReponseAjax  {
  val gen = new Random()
  val contenu = List("J'aime le cours LOG4420",
                     "J'aime le professeur de LOG4420",
                     "J'aime le professeur de INF4215",
                     "Je vais réussir le cours LOG4420",
                     "Je serai bientôt ingénieur")

  // Pour choisir une chaîne au hasard
  def select() = contenu(gen.nextInt(contenu.length))

  def render = {
    "button" #> SHtml.ajaxButton(Text("Changer"),
        ()=>{ Run("$('.contenu > p').text(\"" + select() + "\"); ") })
  }
}
```

Expliquez ce qui sera affiché par le navigateur et comment se comporte cette application web lorsqu'on interagit avec elle. Plus précisément, indiquez les actions que l'utilisateur peut effectuer, leurs effets, ainsi que les communications entre le navigateur et le serveur.

Lorsque la page est chargée, un bouton *Changer* apparaîtra, avec en dessous un message pris au hasard parmi la liste *contenu*. Chaque fois qu'on clique sur le bouton, une connexion asynchrone est effectuée avec le serveur, qui renvoie alors du code jQuery qui sera exécuté par le navigateur. Ce code remplace le message par un autre choisi aléatoirement. Le texte remplacé est celui se trouvant dans la balise <p> qui ser trouve immédiatement à l'intérieur de l'élément dont la classe est *contenu*

4 Serveur (4 points)

a) (1 point) Dans l'approche MVC pour un serveur web, on utilise des gabarits (templates) pour la vue. En quoi cela constitue-t-il un avantage par rapport à la technologie CGI ?

Cela permet de mieux séparer la vue du modèle. Alors que dans l'approche CGI, c'est l'application qui a la responsabilité de tout générer, avec les gabarits utilisé par un MVC, on peut utiliser du code HTML déjà presque formé et laisser la plateforme gérer l'interaction entre cette vue et le code qui permet d'y ajouter le contenu dynamique.

b) (1,5 point) Soit l'URI `http://exemple.com/index`. Supposons que nous utilisons la plateforme Lift pour y associer la vue suivante :

```
<!doctype html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=UTF-8" >
</head>
<body class="lift:content_id=real_content">
  <div data-lift="surround?with=default;at=content" id="real_content">
    <h1> Tâches </h1>
    <ul data-lift="TaskSnippet.showTaskList">
      <li>
        <form data-lift="TaskSnippet.deleteTask?form=post">
          <span> Tâche </span>
          <input type="hidden" name="taskId" value="0">
          <input type="submit" value="Delete">
        </form>
      </li>
    </ul>
    <h1>Ajouter une nouvelle tâche</h1>
    <form data-lift="TaskSnippet.newTask?form=post">
      Tâche: <input name="name"><br>
      <input type="submit" value="Create">
    </form>
  </div>
</body>
</html>
```

Énumérez les étapes du traitement effectué par Lift pour prendre la requête et générer la réponse HTML. Remarque : votre réponse devrait contenir les termes *sitemap*, *default.html* et *snippet*. Veuillez noter les éléments importants qui sont démarqués en gras.

Dans un premier temps, Lift vérifie avec le sitemap si l'URL est valide. Si c'est le cas, la vue est extraite et l'élément *div* se trouvant dans l'élément *body* est transformé, puis inclus dans le gabarit *default.html*. Le premier élément *ul* est traité par le snippet *showTaskList*, puis le résultat obtenu est encore une fois traité, en passant les formulaires qu'il contient au snippet *deleteTask*. Finalement, le formulaire qui constitue les dernier membre du *div* est traité par le snippet *newTask*.

c) (1,5 point) Soit le modèle suivant en Scala :

```
import net.liftweb.util.Helpers._
```

```
case class Client(nom: String, cie: String, siteWeb: String)
val clients= List(
  Client("Martin Odersky", "Typesafe Inc.", "http://www.typesafe.com"),
  Client("Steve Ballmer", "Microsoft Inc.", "http://www.microsoft.com"),
  Client("Bill Venners", "Artima Inc.", "http://www.artima.com/"))
```

Soit maintenant la vue suivante en HTML :

```
<ul>
   <li class="client">
      <div class="nom">Bob Foo</div>
      <a class="cie" href="ecma.com">ECMA inc.</a>
   </li>
</ul>
```

Écrivez le transformateur que vous incluerez dans votre snippet pour obtenir le fichier HTML suivant :

```
<ul>
   <li class="client">
      <div class="nom">Martin Odersky</div>
      <a href="http://www.typesafe.com" class="cie">Typesafe Inc.</a>
   </li>
   <li class="client">
      <div class="nom">Steve Ballmer</div>
      <a href="http://www.microsoft.com" class="cie">Microsoft Inc.</a>
   </li>
   <li class="client">
      <div class="nom">Bill Venners</div>
      <a href="http://www.artima.com/" class="cie">Artima Inc.</a>
   </li>
</ul>
```

```
".client *" #> (clients.map( client =>
      { case Client(nom,cie,site) =>
          (".nom *" #> nom &
           ".cie *" #> cie &
           "#.cie [href]" #> site) }))
```

ou

```
".client *" #> clients.map { client=>
       ".nom *" #> client.nom &
       ".cie *" #> client.cie &
       ".cie [href]" #> client.siteWeb
       }
```

5 Services web (2 points)

a) (1 point) Le service web que vous avez développé dans votre travail pratique est de niveau 2, selon le modèle de Richardson. Expliquez pourquoi et, en vous servant d'un exemple, dites ce qu'il faudrait changer pour qu'il soit de niveau 3 ?

Parce qu'il utilise les verbes du protocole HTTP. Il faudrait qu'il retourne des liens pour poursuivre les interactions. Par exemple. lorsqu'on fait un GET avec `www.polymtl.ca/cheminement`, on obtient un document XML qui contient des liens avec les cheminments de la forme suivante :

```
<link rel="item" uri="www.polymtl.ca/cheminement/50ad9d5eef86379d67fce332"/>
```

b) (1 point) Citez un avantage important d'un service web REST par rapport à un service utilisant le modèle SOAP, et illustrez cet avantage avec un exemple.

On peut profiter des performances dûes aux caches. Par exemple, une requête GET ne sera pas nécessairement acheminée au service web si elle se trouve dans la cache d'un proxy.

6 NoSQL (2 points)

Soit un site web pour la gestion des charges de cours des professeurs de Polytechnique. Représenté en JSON, le modèle de données ressemblerait à ceci :

```
{
  "Automne 2012": {
     "Michel Gagnon": [ {"LOG440":76}, {"INF6410":12} ],
     "Miguel Winner": [{"INF6432":245},{"LOG8001":4},{"INF1212":45}]
  },

  "Hiver 2013": {
     "Michel Gagnon": [{"INF4215":59}]
     "Barack Obama": [{"LOG8001":13}]
  }
}
```

On remarque que pour chaque session, on a la liste des professeurs impliqués et la liste des cours dont ils ont eu la charge pour cette session. Pour chaque cours, on indique le nombre d'étudiants inscrits.

a) (1 point) Maria, qui a suivi le cours LOG4420, croit qu'une base de données implémentée avec MongoDB serait adéquate pour stocker les données. Paulo lui fait remarquer qu'une base de données relationnelle serait plus adéquate. Fournissez un argument qui pourrait être en faveur du choix de Maria, et un autre en faveur de celui de Paulo.

Si on est appelé à utiliser les données de diverses manières, une base de données relationnelle sera adéquate. Par exemple, si on veut obtenir la liste des cours, ce sera plus efficace qu'avec MongoDB, puisque les cours seront conentus dans une même table, plutôt qu'être dispersés. Par contre, si l'utilisation se limite à obtenir les cours d'un professeur pour une session donnée, et qu'on ne modifie pas

souvent les données, MongoDB sera adéquat, vu son optimisation pour la lecture de données.

b) (1 point) Ana, qui était collègue de Maria dans le cours LOG4420, aime bien les bases de données de type clé-valeur, comme Redis. Comment devra-t-elle adapter le modèle pour pouvoir l'utiliser avec Redis ?

Il faut déterminer ce que seront les clés et ce qui sera stocké. Ici, le plus évident est d'utiliser une clé de la forme `session:professeur`. À chaque clé sera associée une liste de paire (cours,nombre_inscrits).

7 RDF et web sémantique (2 points)

Répondez à exactement deux questions parmi les suivantes :

a) (1 point) Décrivez en RDF la situation suivante : *Le 3 septembre 2010, au restaurant OléOlé, le chef de la Mafia a rencontré le maire de Laval.*

```
[] a :Rencontre ;
   :date "03-09-2012" ;
   :lieu [ a :Restaurant ;
           :nom "OléOlé'' ] ;
   :participant [ :maireDe :Laval ] ;
   :participant [ :chefDe :Mafia ] .
```

b) (1 point) Dessinez le graphe RDF qui correspond à la représentation suivante :

```
@prefix local: <http://polymtl.ca/local#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

local:Josee foaf:knows  [ foaf:knows local:Paul, local:Marie ;
                          local:aReussi _:n1 ] .

local:MichelGagnon local:enseigne _:n1 .
_:n1 a local:Cours ;
     local:titre "LOG4420" .
```

c) (1point) Soit la base de données RDF suivante :

```
@prefix local: <http://polymtl.ca/local#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

local:Josee foaf:knows  local:Henri .
local:Josee foaf:knows  local:Ana .
local:Ana foaf:knows  local:Luiza .
local:Paul  foaf:knows  local:Luiza.

local:Paul local:lives local:Montreal .
local:Josee local:lives local:Montreal .
local:Henri local:lives local:Montreal .
local:Ana local:lives local:Quebec .
local:Luiza local:lives local:Quebec .
```

Quel sera le résultat retourné par la requête SPARQL suivante :

```
PREFIX local: <http://polymtl.ca/local#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?x  ?z WHERE {
   ?x foaf:knows ?z .
   ?x local:lives _:n2 .
   ?z local:lives _:n2 . }
```

local :Josee	local :Henri
local :Ana	local :Luiza

8 Question bonus (0,5 pt)

Qui suis-je ? (Fournissez le nom et le prénom)

Né le 12 septembre 1900 et mort le 1er septembre 1982, je suis un mathématicien et logicien américain. Mes travaux apparentés au lambda-calcul ont posé les bases de la programmation fonctionnelle. Je suis principalement connu pour mon travail sur la logique combinatoire. À titre d'hommage, au moins un langage de programmation porte mon nom.

Haskell Curry

Annexe - DOM

Propriétés et méthodes de l'objet *document* :

getElementById()	Retourne l'élément possédant le ID passé en paramètre.
getElementsByTagName()	Retourne une collection contenant tous les éléments dont la balise correspond à celle passée en paramètre.
getElementsByClassName()	Retourne une collection contenant tous les éléments dont la classe correspond à celle passée en paramètre.
querySelector()	Retourne le premier élément correspondant au sélecteur CSS passé en paramètre.
querySelectorAll()	Retourne une collection contenant tous les éléments correspondant au sélecteur CSS passé en paramètre.
createElement()	Crée un élément correspondant à la balise passée en paramètre.
createTextNode()	Crée un nouveau noeud texte.

Propriétés et méthodes associées à un noeud N du DOM :

innerHTML	Contient tout le contenu HTML inclus dans un noeud.
childNodes	Collection contenant les noeuds fils, incluant les noeuds texte.
children	Collection contenant les noeuds fils, excluant les noeuds texte.
firstChild	Retourne le premier noeud fils
firstElementChild	Retourne le premier fils, en ne considérant pas les noeuds texte.
lastChild	Retourne le dernier noeud fils
lastElementChild	Retourne le dernier fils, en ne considérant pas les noeuds texte.
nextSibling	Retourne le noeud qui a le même parent que N et qui est situé immédiatement après.
nextElementSibling	Retourne le noeud qui a le même parent que N et qui est situé immédiatement après, en ne considérant pas les noeuds texte.
previousSibling	Retourne le noeud qui a le même parent que N et qui est situé immédiatement avant.
previousElementSibling	Retourne le noeud qui a le même parent que N et qui est situé immédiatement avant, en ne considérant pas les noeuds texte.
parentNode	Retourne le noeud parent.
textContent	Retourne tout le contenu textuel d'un noeud (incluant ses descendants).
nodeType	Retourne le type d'un noeud. En particulier : 1 = noeud élément, 2 = noeud attribut, 3 = noeud texte.
style	Permet d'obtenir et de changer le style d'un élément. Exemple : `monElement.style.color = blue`.
className	Permet d'obtenir et de changer la classe d'un élément.
addEventListener(ev,gest,cap)	Associe un gestionnaire d'événement à un événement. Si le troisième argument est $true$ le traitement se fait aussi au cours de la capture (pas seulement lors de la propagation vers le haut (bubbling).
appendChild()	Ajoute le noeud passé en paramètre après tous les noeuds fils de N.
insertBefore(nouv,ref)	Ajoute le noeud $nouv$ juste avant le noeud ref, qui est un noeud fils de N.

Annexe - jQuery

.addClass(classe)	Ajoute une classe à l'élément.
.append(contenu)	Ajoute le contenu à la fin du contenu actuel de l'élément.
.attr(nomAttribut)	Obtient la valeur d'un attribut.
.attr(nom,valeur)	Fixe la valeur d'un attribut.
.blur(gest)	Exécute la fonction *gest* chaque fois que l'élément perd le focus.
.children(sélecteur)	Retourne tous les noeuds fils correspondant au sélecteur (ce dernier est facultatif).
.click(gest)	Exécute la fonction *gest* chaque fois qu'on clique sur l'élément.
.clone()	Fait une copie profonde de l'élément.
.css(prop)	Obtient la valeur d'une propriété de style.
.css(prop,valeur)	Fixe la valeur d'une propriété de style.
.each(fct)	Exécute la fonction pour tous les éléments.
.end()	Ignore la cascade d'appels qui le précède et reprend l'élément original.
.find(selecteur)	Recherche tous les éléments correspondant au sélecteur.
.focus()	Exécute la fonction *gest* chaque fois que l'élément devient le focus.
.hover(gest)	Exécute la fonction *gest* chaque fois que le curseur de la souris se trouve au dessus de l'élément.
.html()	Obtient le contenu HTML.
.html(contenu)	Change le contenu HTML par celui passé en paramètre.
.hover(gestIn,gestOut)	Associe des gestionnaires exécutés quand le curseur de la souris entre et quitte l'espace occupé par l'élément, respectivement.
.off(ev,sélecteur,gest)	Élimine l'association entre un gestionnaire et l'événement, pour tous les éléments représentés par le sélecteur (ce dernier est facultatif).
.on(ev,sélecteur,gest)	Associe un gestionnaire à un événement pour tous les éléments représentés par le sélecteur (ce dernier est facultatif).
.parent(sélecteur)	Retourne le parent, s'il correspond au sélecteur (ce dernier est facultatif).
.ready(gest)	Exécute la fonction *gest* quand le DOM est complètement chargé en mémoire.
.removeClass(classe)	Retire une classe de l'élément.
.text()	Obtient tout le texte contenu dans l'élément.
.text(texte)	Remplace le contenu de l'élément par le texte passé en paramètre.
.toggleClass(classe)	Retire ou ajoute une classe, selon qu'elle est déjà présente ou non.