fixed (right:10pt,top:10pt)

Introduction à l'informatique

Cours 9

kn@lri.fr
http://www.lri.fr/~kn

université
PARIS-SACLAY

Plan

Bref historique d'Internet (1/2)

1959-1968 : ARPA (*Advanced Research Project Agency*) crée un réseau de quelques machines capable de résister à une attaque.

1969 : ARPANET. Interconnexion des ordinateurs centraux des grandes universités et institutions américaines. Première utilisation du concept de paquet d'information.

1970-1982 : Interconnexion avec la Norvège et le Royaume-Uni.

1982 : Généralisation du protocole TCP/IP. Naissance de l'Internet actuel.

Bref historique d'Internet (2/2)

1986 : « Autoroutes de l'information ». Des super-ordinateurs et les premières connexions à fibres optiques sont utilisées pour accélérer le débit d'Internet.

1987-1992 : Apparition des premiers fournisseurs d'accès. Les entreprises se connectent.

1993-2000 : Avènement du Web. Démocratisation du haut débit (vers 2000 pour la France).

2000-? : Explosion des services en ligne, arrivée des réseaux sociaux, internet mobile, *Cloud* (stockage et calcul mutualisés accessible depuis internet).

Internet

- Ensemble de logiciels et protocoles basés sur *TCP/IP*
- Les applications suivent un modèle Client/Serveur
- Un serveur fournit un service:
  - courriel
  - transfert de fichier (ftp)
  - connexion à distance (ssh)
  - Web (http)
- Plusieurs services peuvent être actifs sur la même machine (serveur). Un *port (identifiant numérique)* est associé à chaque service. Sur Internet, un service est identifié par:
  - L'adresse IP de la machine sur laquelle il fonctionne
  - Le numéro de port sur lequel le programme attend les connexions

Démo

On établit une connexion TCP, sur le port 4444 entre notre machine et la machine dont l'IP est `127.0.0.1`.

La commande Unix `netcat` possède deux mode de fonctionnement :

- `netcat -l 4444` : se met en attente de connexion sur le port 4444. Toutes les données reçues sont écrites sur la sortie, toutes les données entrées sont envoyées au client connecté.
- `netcat 127.0.0.1 4444` se connecte à un serveur en TCP sur le port 4444.

World Wide Web (1/2)

- Service de distribution de page *hypertexte*
- Une page *hypertexte* contient des références immédiatement accessibles à d'autres pages (pointeurs ou *liens hypertextes*)
- Les pages sont décrites dans le langage *HTML* (HyperText Markup Language)
- Architecture client/serveur:
  - Les pages sont stockées sur le serveur
  - Les pages sont envoyées au client (navigateur Web) qui en assure le rendu
- Utilise le protocole *HTTP* pour les échanges entre le client et le serveur

World Wide Web (2/2)

Concepts clé:

URL : localisation d'une page Web (« adresse de la page »)

HTTP : protocole de communication entre un client et un serveur Web

HTML : langage de description des pages Web

Évolutions récentes (Web 2.0, internet mobile, *Cloud*, ...)

- Standardisation du contenu multimédia (images, vidéos et sons en *streaming*)
- Contenu interactif avancé (stockage de fichier coté client, rendu 3D, ...)
- Uniformisation de nombreuses extensions *ad-hoc*: HTML5

Plan

Fonctionnement du Web

Côté client

Le navigateur :

- Analyse l'URL demandée
- Obtient l'adresse IP auprès du serveur DNS
- Établit une connexion (potentiellement sécurisée) avec le serveur
- Envoie une *requête HTTP* au serveur
- Récupère la page envoyée par le serveur dans sa *réponse*
- Analyse la page et récupère les éléments référencés : images, sons, ...
- Effectue le traitement du code client
- Met en forme le contenu et l'affiche dans la fenêtre

Côté serveur

- Le serveur attend les connexions sur un port par défaut (80 dans le cas de HTTP)
- À chaque nouvelle connexion le serveur vérifie la validité de la requête :
  - le document demandé existe ?
  - le client est autorisé à accéder au document ?
  - ...
- Le serveur répond à la requête :
  - Exécution de code côté serveur, récupération de données dans une BD, ...
  - Envoi de la page au client

Adressage des documents Web (1/3)

URL : *Uniform Resource Locator* identifie un document sur internet

Une URL se décompose en 3 parties

- *protocole* (comment ?)
- adresse (où ?)
- document (quoi ?)

Syntaxe (simplifiée) :

```
protocole://adresse/document
```

Exemple :

```
http://www.lri.fr/~kn/teach_fr.html
```

Adressage des documents Web (2/3)

On peut aussi préciser un numéro de port, des paramètres et un emplacement :

```
protocole://adresse:port/document?p1=v1&p2=v2#empl
```

Exemple :

```
http://www.youtube.com:80/results?search_query=cat#search-results
```

Le serveur utilise les paramètres passés par le client dans l'URL pour *calculer* le contenu de la page (changer la chaîne « `cat` » ci-dessus et essayer)

Adressage des documents Web (3/3)

La *racine* d'un site Web (ex: *http://www.lri.fr/*) correspond à un répertoire sur le disque du serveur (ex: */var/www*). Le fichier

```
http://www.lri.fr/index.html
```

se trouve à l'emplacement

```
/var/www/index.html
```

Le serveur Web peut aussi effectuer des *réécritures d'adresses* :

```
http://www.lri.fr/~kn/index.html
```

devient

```
/home/kn/public_html/index.html
```

Caractéristiques du protocole HTTP

- Sans connexion permanente:
  - Le client se connecte au serveur, envoie sa requête, se déconnecte
  - Le serveur se connecte au client, envoie sa réponse, se déconnecte
- Indépendant du contenu : permet d'envoyer des documents (hyper) texte, du son, des images, ...
- Sans *état*: chaque paire requête/réponse est indépendante (le serveur ne maintient pas d'information sur le client entre les requêtes)
- Protocole en mode *texte*

Format des messages HTTP

Les messages ont la forme suivante

- Ligne initiale CR LF
- zéro ou plusieurs lignes d'option CR LF
- CR LF
- Corp du message (document envoyé, paramètres de la requête, ...)

- *Requête* la première ligne contient un nom de *méthode* (GET, POST, HEAD, ...), le paramètre de la méthode et la version du protocole
- *Réponse* la version du protocole, le code de la réponse (200, 404, 403, ...) et un message informatif

Démo

- On réutilise le programme `netcat` pour explorer un peu le protocole HTTP
- On utilise d'abord `netcat -l 4444` et on s'y connecte avec un navigateur Web
- En suite, on utilise `netcat www.nsi-premiere.fr 80` pour récupérer une page.

Détail sur HTTP

- *Client :*

```
GET /test.html HTTP/1.1
Host: www.nsi-premiere.fr:80
```

- *Serveur :*

```
HTTP/1.1 200 OK                                          } ← code de retour
Server: nginx/1.10.3 (Ubuntu)
Date: Thu, 13 Jun 2019 18:59:49 GMT
Content-Type: text/html                                  } ← type de contenu
Content-Length: 128                                      } ← longueur du contenu
Last-Modified: Wed, 12 Jun 2019 10:15:01 GMT
Connection: close
ETag: "5d00d0a5-80"
Accept-Ranges: bytes                                     }
<!DOCTYPE html>                                          } ← contenu (128 octets)
<html>
  <head>
    <meta charset="UTF-8" />
    <title>NSI Première : page de test !</title>
  </head>
  <body>
    <h1>Page de test</h1>
    <p>
```

Plan

Éléments de cryptographie (1)

Alice et Bob veulent échanger des données confidentielles.

1. Chiffrement symétrique:
   - Ils se mettent d'accord sur une *clé commune*
   - Alice *chiffre* son message avec la clé et l'envoie à Bob
   - Bob déchiffre le message avec *la clé*

Non sûr (Alice et Bob doivent se mettre d'accord sur une clé en « clair », par email par exemple) ou non pratique (ils doivent se rencontrer physiquement pour échanger la clé).

Efficace: on peut implanter plusieurs algorithmes de chiffrements en utilisant uniquement des opérations logiques de bases (`AND`, `OR`, `XOR`). Il est facile de les implanter sur des puces dédiées (cartes de crédit, processeurs mobiles). Ils sont « sûrs » tant que la clé reste secrète.

Éléments de cryptographie (2)

Alice et Bob veulent échanger des données confidentielles.

2. Chiffrement *asymétrique*:

- Bob crée une *clé publique* $\mathrm{K}^{\mathrm{B}}_{\mathrm{pub}}$ et une *clé secrète* $\mathrm{K}^{\mathrm{B}}_{\mathrm{priv}}$, telle que

$$\forall msg, K^B_{priv}(K^B_{pub}(msg)) = K^B_{pub}(K^B_{priv}(msg)) = msg$$

Bob *diffuse* sa clé publique (sur sa page Web par exemple, ou dans un annuaire de clé) et garde sa clé privée *secrète*.

- Alice *chiffre* son message m avec la *clé publique* de Bob ($\mathrm{K}^{\mathrm{B}}_{\mathrm{pub}}(\mathrm{m})$) et l'envoie à Bob
- Bob déchiffre le message avec sa clé privée: $\mathrm{K}^{\mathrm{B}}_{\mathrm{priv}}(\mathrm{K}^{\mathrm{B}}_{\mathrm{pub}}(\mathrm{m}))=\mathrm{m}$

Sûr et pratique

Peu efficace: repose sur des problèmes mathématiques difficiles. Chiffrer et déchiffrer un message n'est pas réaliste pour des grands messages (vidéo en streaming, requêtes Web, ...).

Éléments de cryptographie (3)

On combine les deux méthodes. (Alice envoie un message à Bob)

- Alice choisit une clé symétrique secrète *s*
- Elle l'envoie à Bob en utilisant la clé publique de ce dernier ($K^B_{pub}(s)$)
- Bob décrypte le message et obtient $s=K^B_{priv}(K^B_{pub}(s))$
- Bob et Alice se sont mis d'accord *de manière sûre* sur une clé commune s! Ils peuvent utiliser un algorithme de chiffrement symétrique pour le reste de la conversation

⇒ Ceci est à la base de protocoles tels que HTTPS

Éléments de cryptographie (4)

Le chiffrement asymétrique permet aussi d'avoir *la preuve* que quelqu'un est bien Bob!

- Alice choisit un message secret aléatoire *m*, sans le divulguer (appelé *challenge*)
- Alice calcule $K^B_{pub}(s)$ et l'envoie à la personne qui prétend être Bob
- Seule la personne qui possède la clé privée de Bob (donc Bob ...) peut déchiffrer le message et renvoyer l'original à Alice.

⇒ Comment garantir que la personne qui a généré les clés au départ est bien Bob ?

Une analogie

La cryptographie asymétrique fonctionne exactement comme l'analogie de la boîte aux lettres. Pourquoi ?

- La clé publique est la boîte aux lettres
- La clé privée est la clé de la boîte aux lettres
- Tout le monde peut «crypter» un message en le glissant dans ma boîte aux lettres
- Une fois le message crypté (*i.e.* dans la boîte) il est difficile de le récupérer sans avoir la clé
- Il est facile pour moi d'ouvrir la boîte avec ma clé

HTTP: protocole texte « en clair »

HTTP est un protocole *texte*, les données ne sont pas chiffrées et *sans identification*

- *Confidentialité* : n'importe qui (avec les privilèges nécessaires) peut lire ce qui transite entre un client et un serveur Web
- *Authenticité* : n'importe qui peut se faire passer pour un serveur Web (attaque *man in the middle*)

Espionnage de connexion

Alice représente le client, Bob le serveur et Eve (*Eavesdropper*) l'attaquante

On suppose que Eve est root sur la machine. Il suffit de lire les paquets qui transitent par la carte réseau (`tcpdump` sous Linux).

- Eve et Alice sont sur la même machine (démo):

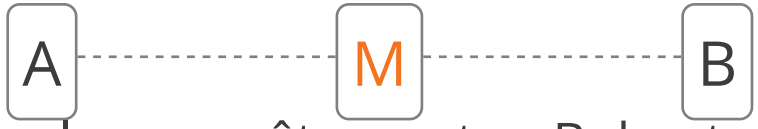
- Fonctionne aussi si Eve est sur une machine se trouvant sur la route entre Alice et Bob:

Ce problème touche tous les protocoles en clair: HTTP, POP, IMAP, FTP, ... Il peut être résolu grâce au *chiffrement* de toute la connexion.

Attaque *Man in the middle*

Mallory se place entre Alice (cliente) et Bob (banque), par exemple au moyen d'un *e-mail* frauduleux en HTML:

1. L'émail contient:
```
<html>
 <body>
     Bonjour,
        veuillez vous connecter à votre banque en cliquant ici:
         <a href=']]><mark>mallory.com</mark>' &gt;<em>www.bob.com</em><![CDATA[<
  </body>
</html>
```
2. Alice, insouciante, clique sur le lien



3. Mallory peut retransmettre les requêtes entre Bob et Alice, en les modifiant au passage. Le problème est causé par un manque d'authentification (Mallory n'a pas a *prouver* à Alice qu'il est Bob)

Solution: HTTPS

HTTP *Secure*

1. Respose sur de la cryptographie asymétrique (pour l'authentification et le partage de clé) et symétrique (pour le chiffrement de connexion)
2. Permet d'authentifier les correspondants et de protéger les données
3. Suppose l'existence de *tiers de confiance* Alice et Bob font confiance à Trent (*Trusted Third Party*)

Bob possède des clés publiques et privées (K^B_{pub} et K^B_{priv}), Trent possède des clés publiques et privées (K^T_{pub} et K^T_{priv})

HTTPS (détail du protocole)

Bob et Trent *se rencontrent*. Trent *signe* la clé publique de Bob en calculant

$$S^B = K^T_{priv}(K^B_{pub})$$

Comme Trent utilise sa clé *privée* on sait que seul Trent a pu générer cette signature. De plus, Trent a *rencontré* Bob donc il *certifie* que la clé K^B_{pub} appartient bien à quelqu'un nommé Bob.

1. Alice (client) veut se connecter à Bob. Bob fournit sa clé publique K^B_{pub} et la signature S^B
2. Alice contacte Trent (en qui elle a confiance) et récupère sa clé publique K^T_{pub}. Elle déchiffre la signature: $K^T_{pub}(S^B)$ et vérifie qu'elle retombe bien sur la clé publique de Bob.
3. Elle peut alors choisir une clé symétrique, la chiffrer avec K^B_{pub} et entamer une communication *chiffrée* et *authentifiée* avec Bob.

Tiers de confiance

Les tiers de confiance sont des entités (états, associations, compagnies privées) qui se chargent de vérifier les clés publiques d'autres entités. C'est une vérification *physique* (documents administratifs, ...).

ssl

Cette erreur s'affiche quand une signature n'est pas conforme ou n'a pas pu être vérifiée

Tiers de confiance

Attaques contre les *authorités de certifications* (tiers de confiance): difficiles, mais pas impossible. Certains tiers de confiance sont douteux (états voyous, compagnie piratées dont les clés privées sont compromises,...)
Attaques d'implémentation (plus probables) : on exploite un bug dans le code des serveurs web ou des navigateurs
Autres faiblesses: HTTPS est en « haut » dans la pile IP (application). On peut donc avoir connaissance du nombre de paquet échangés, des adresses IP des participants, la taille et la fréquence des paquets... (même si on n'en connaît pas le contenu). Cela permet certaines attaques statistiques ou de déni de service (DoS).

Plan

Stateless

Le protocole HTTP est un protocole *sans état*. Lorsque le serveur ferme une connexion TCP après avoir envoyé sa réponse, il « oublie » le client (i.e. rien dans le protoocle n'indique de conserver un historique des requêtes).

Exemple

- Un utilisateur se connecte à un site
- Il saisit sont login/mot de passe et clique « me connecter »
- Le site authentifie l'utilisateur et lui affiche sa page personnelle (et ferme la connexion)
- L'utilisateur recharge la page (nouvelle connexion et requête). Comment pouvoir s'assurer que c'est le même utilisateur ?

Problème Aucun mécanisme dans HTTP (ou dans TCP ou IP) ne permet d'identifier de façon exacte le client, i.e. l'utilisateur qui a fait la première requête.

Cookie

Un *cookie* est un petit paquet d'information, renvoyé par le serveur grâce à l'en-tête `Set-Cookie:`

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Tue, 19 Oct 2021 17:33:53 GMT
Content-Type: text/html
Content-Length: 2
Set-Cookie: foo=superdata42; Max-Age=3600
...
```

Le site crée un cookie appelé `foo`, contenant la valeur `superdata42` et d'une durée de vie de 3600 secondes.

Cette donnée est stockée *par le navigateur et renvoyée pour chaque requête (tant qu'il n'a pas expiré)*.

Utilité ?

On peut faire persister de l'information entre plusieurs requêtes.

Problème : l'information est sous le contrôle du client.

Session HTTP

Une *session* HTTP est un ensemble de paires requêtes/réponses entre *un* client et le serveur. Un état est maintenu côté serveur pour toute la durée de la session. Exemple :

- Un utlisateur se connecte sur un site (début de session)
- Il saisit son login/mot de passe.
- Si valide, le site définit côté serveur un booléen `connecté`
- À chaque chargement de page, le serveur vérifie la valeur du booléen pour voir si l'utilisateur est connecté. Il y a autant de booléens que de sessions.
- Au bout d'un certain temps d'inactivité ou en cas de déconnection explicite, le booléen est supprimé (fin de session)

Implémentation des sessions

On peut implémenter des sessions en utilisant des cookies.

On suppose que le serveur maintient un dictionnaire de dictionnaires appelé `S`.

1. Lorsque le client se connecte, regarder la valeur d'un cookie particulier (par exemple `MySessionID`)
2. Si ce cookie n'existe pas, on choisit un identifiant aléatoire (*i*), et on crée une nouvelle entrée : `S[i] = {}`
3. On envoie au client l'en-tête `Set-Cookie: MySessionID=`*i*`;Max-Age=600`.
4. Lorsque le client se reconnecte et fournit le cookie, on va chercher `S[i]` et le serveur peut alors y stocker des données propres au client.
5. Si le client ne se connecte pas au bout de 10 minutes, le navigateur détruit le cookie, et on reprend en 1.

Considérations de sécurité pour les sessions

- L'ID de session doit être difficile à deviner/énumérer (⇒ entier aléatoire de taille conséquente >= 128 bits)
- Les cookies peuvent être *signés* avec une clé du serveur (⇒ vérifie que le client ne modifie pas un cookie)
- Les entrées inutilisées dans la table de session doivent être régulièrement vidée. Une adresse IP effectuant trop de connexions doit être bannie temporairement (DOS).

Plan

HTML

HyperText Markup Language : langage de mise en forme de documents hypertextes (texte + liens vers d'autres documents). Développé au CERN en 1989.
1991 : premier navigateur en mode texte
1993 : premier navigateur graphique (mosaic) développé au NCSA (National Center for Supercomputing Applications)

Document HTML

- est un document *semi-structuré*
- dont la structure est donnée par des *balises*

Exemple	**Rendu par défaut**
`Un texte <b>en gras</b>`	Un texte **en gras**
`<a href="http://www.u-psud.fr">Un lien</a>`	Un lien
`<ul >   <li>Premièrement</li>   <li>Deuxièmement</li> </ul>`	◆ Premièrement ◆ Deuxièmement

On dit que

`<toto>` est une balise *ouvrante* et **`</toto>`** une balise *fermante*. On peut écrire **`<toto/>`** comme raccourci pour **`<toto></toto>`**.

Historique du langage HTML

1973 : GML, Generalised Markup Language développé chez IBM. Introduction de la notion de balise.

1980 : SGML, Standardised GML, adopté par l'ISO

1989 : HTML, basé sur SGML. Plusieurs entreprises (microsoft, netscape, ...) interprètent le standard de manière différente

1996 : XML, eXtensible Markup Language norme pour les documents semi-structurés (SGML simplifié)

2000 : XHTML, version de HTML suivant les conventions XML

2008 : Première proposition pour le nouveau standard, HTML5

2014 : Standardisation de HTML 5.0

2017 : Standardisation de HTML 5.2

Rôle d'HTML

Séparer la *structure* du document de son *rendu*. La structure donne une *sémantique* au document :

- ceci est un titre
- ceci est un paragraphe
- ceci est un ensemble de caractères importants

Cela permet au navigateur d'assurer un rendu en fonction de la sémantique. Il existe différents types de rendus:

- graphique interactif (Chrome, Firefox, Internet Explorer, ...)
- texte interactif (Lynx, navigateur en mode texte)
- graphique statique (par ex: sur livre électronique)
- rendu sur papier
- graphique pour petit écran (terminal mobile)

Exemple de document

```
  <!DOCTYPE html>
<html>

  <head>
    <title>Un titre</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <h1>Titre de section</h1>
    <p> premier paragraphe de texte. On met
    un <a href="http://www.lri.fr">lien</a> ici.
    </p>
    <!-- on peut aussi mettre des commentaires -->
  </body>

</html>
```

Structure d'un document HTML

Pour être *valide* un document HTML contient au moins les balises suivantes :

- Une balise `html` qui est la racine (elle englobe toutes les autres balises). La balise `html` contient deux balises filles: `head` et `body`
- La balise `head` représente l'en-tête du document. Elle peut contenir diverses informations (feuilles de styles, titre, encodage de caractères, ...). La seule balise obligatoire dans `head` est le titre (`title`). C'est le texte qui est affiché dans la barre de fenêtre du navigateur ou dans l'onglet.
- la balise `body` représente le contenu de la page. On y trouve diverses balises (`div`, `p`, `table`, ...) qui formatent le contenu de la page

Titres

Les balises `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, permettent de créer des titres de section, sous-section , sous-sous-section ,...

Titre de niveau 1

Titre de niveau 2

Titre de niveau 3

Titre de niveau 4

Titre de niveau 5

Paragraphes

Des paragraphes de textes sont introduits avec les balises `<p>`. Par défaut chaque paragraphe implique un retour à la ligne:

```
<p>Lorem ipsum      dolor sit amet, consectetur
 adipiscing elit, sed do eiusmod tempor incididunt ut labore et
 dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
 exercitation ullamc</p>
<p>Nouveau paragraphe</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamc

Nouveau paragraphe

Remarque : par défaut, les espaces, retour à la ligne, ... sont ignorés et le texte est reformaté pour remplir la largeur de la page.

Mise en forme du texte

Les balises `<b>` (*bold*, gras), `<i>` (*italic*, italique), `<u>` (*underlined*, souligné) `<em>` (*emphasis*, important) et beaucoup d'autres permettent de décorer le texte.

```
<p><b>Lorem ipsum dolor</b> sit amet, consectetur
  adipiscing elit, sed do eiusmod tempor incididunt ut labore et
  dolore magna aliqua. <u>Ut enim ad minim veniam</u>, <em>quis</em> nostrud
  exercitation ullamc</p>
 <p><i>Nouveau</i> paragraphe</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, *quis* nostrud exercitation ullamc

Nouveau paragraphe

Tableaux

On peut formater des tables en utilisant :

- La balise `<table>` pour délimiter la table
- La balise `<tr>` pour délimiter une ligne de la table
- La balise `<th>` pour délimiter une tête de colonne
- La balise `<td>` pour délimiter une case
- L'attribut `colspan` permet de fusionner des colones

```
<table>
 <tr> <th>Nom</th> <th>Prénom</th> <th>Note 1</th> <th>Note 2</th></tr>
 <tr> <td>Foo</td>  <td>Bar</td>   <td> 15</td> <td>12</td> </tr>
 <tr> <td>Doe </td> <td>Jonh</td>  <td colspan="2">Absent</td></tr>
</table>
```

Nom	Prénom	Note 1	Note 2
Foo	Bar	15	12
Doe	Jonh	Absent	

Les espaces et retours à la ligne ne sont là que pour rendre le code lisible !

Listes

On peut créer des listes énumérées (avec `<ol>`, *ordered list*) ou non énumérées (avec `<ul>`, *unordered list*). Chaque ligne est limitée par une balise `<li>` (*list item*)

```
<ul>
 <li> Un élément </li>
 <li> <ol> <li> Un autre élément </li>
           <li> <ol> <li> Un sous-élément</li>
                     <li> Un autre sous-élément</li>
               </ol>
          </li>
     </ol>
 <li>Le dernier</li>
</ul>
```

- Un élément
- 1. Un autre élément
  2. 1. Un sous-élément
     2. Un autre sous-élément

Liens hyper-texte

On peut faire référence à une autre ressource en utilisant un lien hyper-texte (balise `<a/>` et son attribut href). La cible du lien peut être absolue (une URL complète avec le protocole, par exemple `https://www.lri.fr`) ou relative (par exemple `foo.html`). Si l'URL est relative, le chemin est substitué à la dernière composante de l'URL de la page courante. Si l'URL commence par un # elle référence, l'attribut `id` d'un élément de la page:

```
<a href="https://www.lri.fr">Le LRI</a>
<a href="../../../index.html">Un lien</a>
<a href="#foo">On va vers le titre</a>
…
<h1 id="foo">Le titre</h1>
```

Le LRI Un lien On va vers le titre …

Remarques générales

- On n'a normalement pas le droit de mettre n'importe quel élément n'importe où (i.e. pas de `<li>` tout seul)
- Il existe une spécification précise de HTML 5 (plusieurs dizaines de pages uniquement pour les balises)
- Il existe aussi des validateurs, il faut les utiliser le plus possible
- De manière générale, les espaces sont ignorés, on prendra donc bien soit de les utiliser judicieusement pour rendre le code de la page lisible
- Tous les éléments ont un style (moche) par défaut. On verra comment modifier ce style grâce à des propriétés CSS.

Plan

Cascading Style Sheets (CSS)

CSS : Langage permettant de décrire le *style graphique* d'une page HTML

On peut appliquer un style CSS :

- À un élément en utilisant *l'attribut* `style`
- À une page en utilisant l'élément **`<style>...</style>`** dans l'en-tête du document

(dans la balise **`<head>...</head>`**).

- À un ensemble de pages en référençant un fichier de style dans chacune des pages

L'attribut `style`

```
<a href="http://www.lri.fr" style="color:red">Un lien</a>
```

Apperçu:

Un lien

Inconvénients :

- il faut copier l'attribut style pour tous les liens de la page
- modification de tous les éléments difficiles

L'élément `style`

```
<html>
    <head>
       <title>…</title>
       <style>
        a { color: red; }
      </style>
    </head>
    <body>
      <a href="…">Lien 1</a> <a href="…">Lien 2</a>
    </body>
  </html>
```

Apperçu :

Lien 1 Lien 2

Inconvénient : local à une page

Fichier `.css` séparé

Fichier `style.css`:

```
a { color: red; }
```

Fichier `test.html`:

```
<html>
  <head>
  …
  <link href="style.css" type="text/css" rel="stylesheet" />
  </head>
  …
</html>
```

Modifications & déploiement aisés

Syntaxe

Une *propriété* CSS est définie en utilisant la syntaxe:

```
nom_prop : val_prop ;
```

- Si on utilise l'attribut `style` d'un élément:

```
<a href="…" style="color:red;border-style:solid;border:1pt;">Lien 1</a>
```

- Si on utilise un fichier `.css` ou une feuille de style:

```
a {
        color : red;
        border-style: solid;
        border: 1pt;
  }
```

Plan

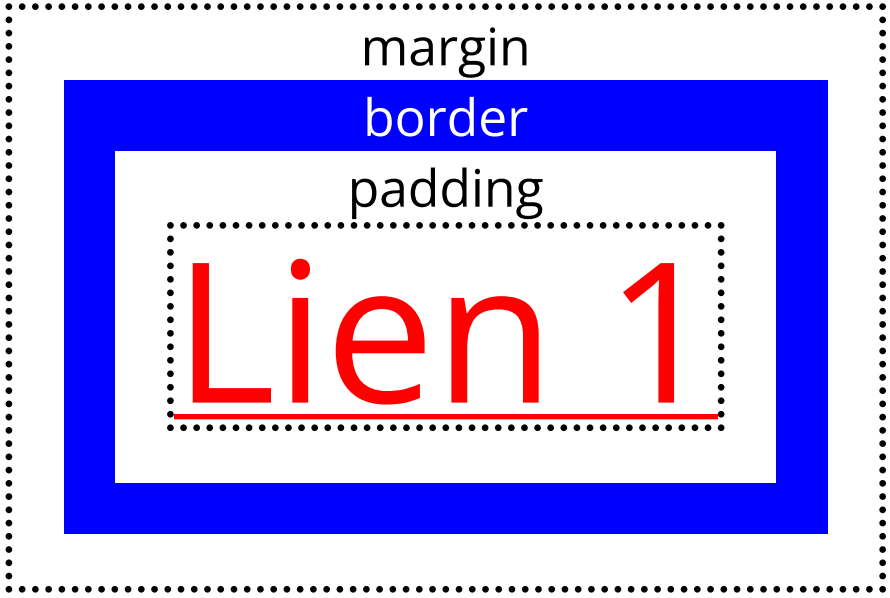
Unités de longueur

CSS permet de spécifier des longueurs comme valeurs de certaines propriétés (position et taille des éléments, épaisseur des bordures, ...). Les longueurs *doivent* comporter une unité. Les unités reconnues sont:

px : pixel
in : pouce (2,54cm)
cm : centimètre
mm : millimètre
pt : point (1/72ème de pouce, 0,35mm)
pc : pica (12 points)
em : facteur de la largeur d'un caractère de la police courante
ex : facteur de la hauteur d'un caractère « x » de la police courante
% : pourcentage d'une valeur particulière (définie par propriété)
vh : *viewport height* (% de la hauteur de la partie visible de la page) CSS3
vw : *viewport width* (% de la largeur de la partie visible de la page) CSS3
vmin : plus petite valeur entre vw et vh CSS3
vmax : plus grande valeur entre vw et vh CSS3

Boîte

Chaque élément de la page HTML possède une *boîte rectangulaire* qui délimite le contenu de l'élément:



La zone entre le contenu et la bordure s'appelle le *padding* (« remplissage »). La zone autour de la bordure s'appelle *margin* (« marge »).

Marge, bordure, ajustement

On peut spécifier jusqu'à 4 valeurs:

- 1 valeur: toutes les dimensions égales à cette valeur
- 2 valeurs: haut et bas égal à la première valeur, gauche et droite égale à la deuxième
- 3 valeurs: haut à la première valeur, gauche et droite égale à la deuxième, bas égal à la troisième
- 4 valeurs: haut, droit, bas, gauche

```
span {
    padding:10pt 20pt 5pt 0pt;
    margin:10pt 5pt;
    border-width:3pt;
    border-color:red blue green;
    border-style:solid dotted;
}
```

Du  dans une boite

Calcul de la taille d'une boîte CSS3

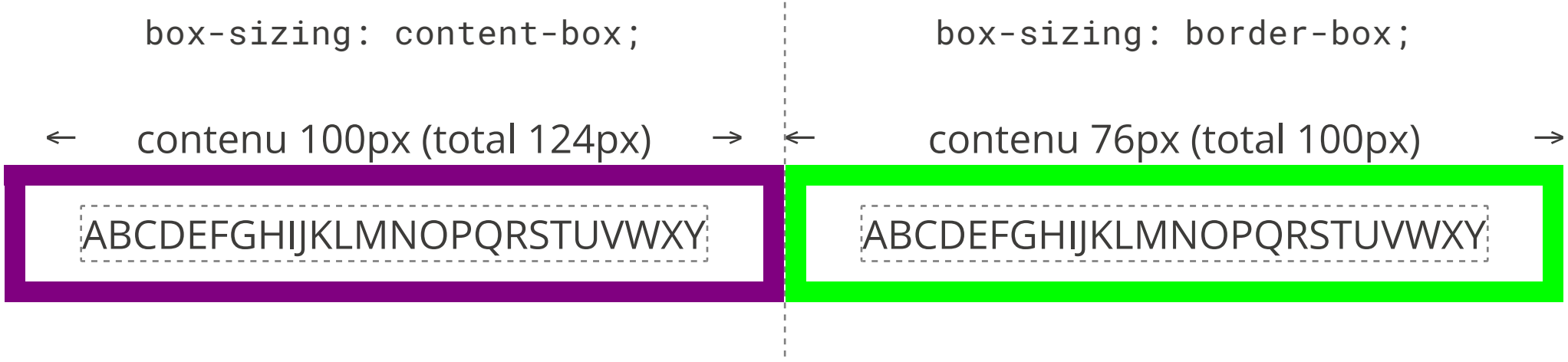
L'attribut CSS `box-sizing` permet de spécifier le mode de calcul de la taille d'une boîte. Deux valeurs sont possible :

`content-box`
: (valeur par défaut) signifie que la taille d'une boîte (telle que donnée par `width` ou `height`) est uniquement celle de son contenu.

`border-box`
: signifie que la taille d'une boîte (telle que donnée par `width` ou `height`) est uniquement celle de son contenu, de l'ajustement et des bordures.

la marge n'est jamais prise en compte dans le calcul de la taille.

Exemple : `border: 2px; padding: 10px; width: 100px;`



Modes d'affichage

La propriété *display* contrôle le mode d'affichage d'un élément:

none : l'élément n'est pas dessiné et n'occupe pas d'espace

inline : l'élément est placé sur la ligne courante, dans le flot de texte. La taille du contenu (avec les marges, ajustements et bordures) dicte la taille de la boîte, `height` et `width` sont ignorés (**`<i>, <b>, <span>, <em>, …`** sont *inline* par défaut).

block : l'élément est placé seul sur sa ligne. La taille est calculée automatiquement mais peut être modifiée par `width` et `height` (**`<div>, <h1>, <p>, …`** sont *block* par défaut)

inline-block : positionné comme *inline* mais la taille peut être modifiée comme pour *block*

Modes d'affichage (exemples)

```
a { display: inline; … }
```

Le lien 1, le lien 2 et le lien 3.

```
a { display: none; … }
```

Le , le et le .

```
a { display: block; … }
```

Le
lien 1
, le
lien 2
et le
lien 3
.

Positionnement

Le type de positionnement est donné par la propriété *position*

static : positionnement « automatique »

fixed : positionnement par rapport à la fenêtre du navigateur (la boîte est supprimée du flot)

relative : positionnement « relatif » par rapport à la position normale

absolute : positionnement « absolu » par rapport à l'ancêtre le plus proche qui n'est pas *static*

Pour `fixed`, `relative` et `absolute`, les propriétés `top`, `bottom`, `left` et `right` dénotent les décalages respectifs.

Positionnement (exemple)

```
a { position: static;
  … }
a { position: fixed;
  right:10pt;
  top: 10pt;

}
```

```
a { position: relative;
  left: 10pt;
  bottom: -5pt;
  … }
a { position:absolute;
  right:0pt;
  bottom: 10pt;
}
```

```
<ul style="position:relative;"><li>…</li> …</ul>
```

- Positionnement static
- Positionnement
- Positionnement relative (left:10pt,bottom:-5pt)
- Positionnement absolute (right:10pt,bottom:10pt)

Gestion du débordement

L'attribut `overflow` permet de gérer le débordement. Il peut prendre les valeurs `visible`, `hidden` et `auto` :

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor

Couleurs

Les couleurs peuvent être données:

- par nom symbolique: red, blue, purple, ...
- en hexadécimal: `#xxyyzz`, avec `00 ≤ xx,yy,zz ≤ ff`
- en décimal: `rgb(x, y, z)`, avec `0 ≤ x,y,z ≤ 255`
- en décimal avec transparence: `rgba(x, y, z, a)`, avec `0 ≤ x,y,z ≤ 255` et `0 ≤ a ≤ 1`

CSS3

On peut définir la couleur de fond d'une boîte avec la propriété `background` et la couleur du texte avec la propriété `color`

Propriétés du texte

Certaines propriétés permettent d'alterer le rendu du texte d'un élément

direction :	`ltr` ou `rtl` (orientation du texte)
text-transform :	`capitalize`, `uppercase`, `lowercase`
text-decoration :	`underline`, `overline`, `line-through`
text-align :	`left`, `right`, `center`, `justify`
text-indent :	`longueur du retrait de paragraphe`
vertical-align :	`alignement vertical par rapport à la ligne de texte baseline, top, bottom, middle ou une longueur.`

Propriétés de la police

font-family : liste de nom de polices séparées par des virgules (`Helvetica, sans, "Times New Roman"`)

font-style : `normal`, `italic`

font-weight : `normal`, `lighter`, `bold`, `bolder`

font-size : soit une longueur soit `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`

On peut aussi spécifier un descripteur de police CSS3

```
@font-face {
  font-family: Toto;
  src: url(toto.ttf);
}
a { font-family: Toto; }
```

Plan

Sélectionneurs

On peut sélectionner finement les éléments auxquels un style s'applique :

`*` :	tous les éléments
`x` :	tous les éléments dont la balise est `x`
`.foo` :	tous les éléments dont l'attribut `class` contient `foo`
`x.foo` :	(sans espace) tous les éléments dont l'attribut `class` contient `foo` et dont la balise est `x`
`#foo` :	l'élément dont l'attribut `id` vaut `foo` (les `id` doivent être uniques)
`X Y` :	tous les éléments selectionnés par `Y` qui sont des descendants des éléments sélectionnés par `X`
`X, Y` :	tous les éléments selectionnés par `X` ou par `Y`
`X > Y` :	tous les éléments dont selectionné par `Y` qui sont des fils d'éléments sélectionnés par `X`
`a:visited` :	les liens déjà visités
`a:link` :	les liens non visités
`X:hover` :	élément selectionné par `X` et survollé par la souris

La spécification CSS3 en définit beaucoup d'autres ...

Attributs `id` et `class`

On veut souvent pouvoir appliquer un style à un unique élément d'une page, ou a un groupe d'éléments bien défini. On utilise pour celà l'attribut `id` de l'élément ou l'attribut class commun à plusieurs éléments:

Fichier CSS:

```
#toto123  {  background:red; }
.bluetext { color: blue; }
.border   { border: 1pt solid green; }
```

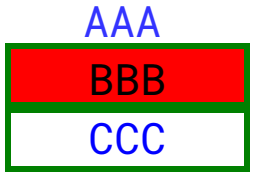
L'élément d'id toto123 a un fond rouge.

Les éléments de classe bluetext sont bleus

Les éléments de classe border une bordure verte

Fichier HTML

```
<html>
 <head> ... </head>
 <body>
  <div class="bluetext"> AAA </div>
  <div id="toto123" class="border"> BBB </div>
  <div class="border bluetext"> CCC </div>
 </body>
</html>
```



Composition de sélectionneurs

Les sélectionneurs CSS peuvent être composés afin de créer des chemins de sélection :

```
div li a  { … } /* tous les a descendants d'un li descendants d'un div */

#id100 > p .foo { … } /* tous les éléments de class foo se trouvant
                         à n'importe quelle profondeur sous un p
                         lui-même étant un sous-élément direct
                         de l'élément d'id id100 */

p.bar { … } /* toutes les balises p qui ont la classe bar */

p  .bar { … } /* toutes la balises ayant la classe bar placées
                 sous une balise p (attention à l'espace !) */

a, li * /* tous les a ou tous les éléments placés sous un li */
```

Cascade ? (priorités)

Que se passe-t-il quand un éléments est sélectionnés par plusieurs sélectionneurs ?

- Les propriétés présentent exclusivement dans l'un ou l'autre sont appliquées
- Pour les propriétés présentes dans plusieurs sélectionneurs (ex: `color:blue;` et `color:red;`) :

1. Les propriétés des attributs `style` ont une priorité plus forte que celles de l'élément `<style >` de l'en-tête, qui a lui même une priorité plus forte que les feuilles de styles `.css` référencées.
2. On prend ensuite pour chaque règle le triplet :
   `(nombre d'id (#foo), nombre de classes (.bar), nombre de balises)`
   On sélectionne les règles ayant la plus forte valeur, comparé composantes par composantes (ordre lexicographique). S'il reste plusieurs règles possibles, on prend la dernière déclaration dans l'ordre du fichier.

Exemple de sélectionneurs ambigus


```
*  { color: blue; }                  /* score (0, 0, 0) */

#id101 li b    { color: red; }       /* score (1, 0, 2) */

#id101 ol *  b { color: green; }     /* score (1, 0, 2) */

.class ol li b { color: pink; }      /* score (0, 1, 3) */
```

Le score le plus élevé est `(1, 0, 2)`. Il y a deux sélectionneurs qui ont ce score, on choisit le dernier dans l'ordre du fichier donc le texte des éléments sélectionnés sera vert.

exemple de pseudo-classes: menu dépliable

```
<ul class="menu">
  <li>Entrée 1
     <ul class="sous-menu"> <li>Sous-entrée 1.1 </li>
       <li>Sous-entrée 1.2 </li>
       <li>Sous-entrée 1.3 </li>
     </ul>
  </li>
  <li>Entrée 2
     <ul class="sous-menu"> <li>Sous-entrée 2.1 </li>
       <li>Sous-entrée 2.2 </li>
       <li>Sous-entrée 2.3 </li>
     </ul>
  </li>
</ul>
```

Analyse

Pour que le ménu soit « dépliable » lors du survol de la souris, on souhaite que :

- Par défaut, les éléments de sous-menu soient cachés (`display : none`)
- Les éléments se trouvant sous un élément survolé (`hover`) soient visibles (`display : block`)

exemple : menu dépliable (démo)