

# UAA12 Introduction aux sites web dynamiques

## 1 Bibliographie

### 1.1 Cours Web

- [1] Prérequis : UAA3, UAA6 web statique
- [2] Chaîne Youtube de Pierre Giraud (et cours pdf dans le drive) HTML et CSS  
<https://www.youtube.com/playlist?list=PLwLsbqvBIImHG5yeUCXJ1aqNMgUKi1NK3>
- [3] Lycée des Flandres, IHM sur le Web - cours dans le drive ou sous :  
[https://qkzk.xyz/docs/nsi/cours\\_premiere/ihm\\_web](https://qkzk.xyz/docs/nsi/cours_premiere/ihm_web)
- [4] Cours d'Openclassroom Site Web statiques et dynamiques - pdf p9 dans le drive ou sous (remplacer php par flask à la lecture du cours):  
<https://openclassrooms.com/fr/courses/918836-concevez-votre-site-web-avec-php-et-mysql>

### 1.2 Flask

- [10] Prérequis: UAA5 Python
- [11] Flask - site officiel  
<https://flask.palletsprojects.com>
- [12] Jinja - site officiel (langage de programmation dans template html)  
<https://jinja.palletsprojects.com>
- [13] Lycée des Flandres, Formulaire : utilisation de Flask - tp de découverte, pdf dans le drive ou sous :  
[https://qkzk.xyz/docs/nsi/cours\\_premiere/ihm\\_web/flask](https://qkzk.xyz/docs/nsi/cours_premiere/ihm_web/flask)
- [14] Les maths du Yeti, Intéraction client - serveur Flask : tp de découverte en complément de [1]  
<https://lesmathsduyeti.fr/fr/lycee/nsi-premiere/interactions-client-serveur-flask>
- [15] Learn Python Programming - Flask tutorial (en anglais)  
<https://pythonbasics.org>
- [16] Un des sites de référence de Flask  
<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- [17] TP - Créer une appli web (css, sql, javascript)  
<https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3-fr>
- [18] Autre bref tutorial pour débiter avec Flask  
[http://www.xavierdupre.fr/app/ensae\\_teaching\\_cs/helpsphinx/notebooks/TD2A\\_eco\\_debuter\\_flask.html](http://www.xavierdupre.fr/app/ensae_teaching_cs/helpsphinx/notebooks/TD2A_eco_debuter_flask.html)

### 1.3 SQL

- [20] Prérequis : UAA7 Bases de données - Cours et tp en SQLite3
- [21] Description des fonctions SQLite3  
<https://www.sqlitetutorial.net/sqlite-functions>
- [22] Cours en ligne : Introduction aux BD, SQL  
<https://www.pierre-giraud.com/php-mysql-apprendre-coder-cours/introduction-sql-mysql>
- [23] Requêtes préparées en sqlite

## 1.4 Javascript

- [30] Les maths du Yeti, Interaction client - serveur Flask : tp de découverte en complément de [1]  
<https://lesmathsduyeti.fr/fr/lycee/nsi-premiere/interactions-entre-lhomme-et-la-machine-sur-le-web/decouverte-javascript>
- [31] Cours d'Openclassroom Ecrivez du javascript pour le web - pdf dans le drive ou sous:  
<https://openclassrooms.com/fr/courses/5543061-ecrivez-du-javascript-pour-le-web?archived-source=1916641>
- [32] Ressources en ligne par les développeurs, pour les développeurs dont les documents de référence html / css / javascript  
<https://developer.mozilla.org/fr>
- [33] David Roche, Découverte de javascript : tp dont le pdf est dans le drive (lycée)
- [34] Lycée La Martinière Didero : tp de découverte de javascript (lycée) :  
[http://portail.lyc-la-martiniere-diderot.ac-lyon.fr/srv1/html/cours\\_js\\_nsi/accueil\\_cours\\_js\\_nsi.html](http://portail.lyc-la-martiniere-diderot.ac-lyon.fr/srv1/html/cours_js_nsi/accueil_cours_js_nsi.html)
- [35] Livre en ligne (en anglais)  
<https://javascript.info>
- [36] TP Introduction à HTML5, CSS3 et JS (niveau universitaire, avec des références de sites web)  
<https://diu-eil.univ-lyon1.fr/bloc1/bloc1v2/WEB/TP.html#partie-programmation-en-javascript>
- [37] Chaîne Youtube de Pierre Giraud (et cours pdf dans le drive) JS  
<https://www.youtube.com/playlist?list=PLwLsbqvBIImFB8AuT6ENIg-s87ys4yGWI>

## 2 Outils et installation

### 2.1 Sites web

- [40 ] Editeur en ligne html / css / JavaScript : JS Fiddle  
<https://jsfiddle.net>
- [41] Les maths du Yeti, Interaction client - serveur Flask : instructions d'installation de Flask sous Windows  
<https://lesmathsduyeti.fr/fr/lycee/nsi-premiere/interactions-client-serveur-flask>
- [42] Générateur d'arbre html  
<https://yoksel.github.io/html-tree/en>
- [43] Tutoriel de configuration de vs code pour flask  
<https://code.visualstudio.com/docs/python/tutorial-flask>
- [44] Tutoriel video de configuration de vs code pour sqlite  
<https://grafikart.fr/tutoriels/sqlite-vscode-1981>

### 2.2 Installations sous linux

#### 1. Python

```
apt-get update; apt-get install idle3
```

#### 2. Environnement virtuel pour Flask

```
apt-get update; apt-get install python3-pip  
python3 -m pip install virtualenv
```

### 3. SQLite3

```
apt-get update; apt-get install sqlite3
```

### 4. Visual Studio Code

```
apt-get install wget gpg  
wget -qO- https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > packages.microsoft.gpg  
install -o root -g root -m 644 packages.microsoft.gpg /etc/apt/trusted.gpg.d/  
sh -c 'echo "deb [arch=amd64,arm64,armhf signed-by=/etc/apt/trusted.gpg.d/packages.microsoft.gpg]  
https://packages.microsoft.com/repos/code stable main" > /etc/apt/sources.list.d/vscode.list'  
rm -f packages.microsoft.gpg  
apt install apt-transport-https  
apt update  
apt install code
```

## 2.3 Installations sous windows

### 1. Python

- ✓ Vas sous [www.python.org](https://www.python.org) et télécharge la dernière version de python pour Windows
- ✓ Installe python (pense à sélectionner l'onglet « add python to PATH »)
- ✓ Lance idle

### 2. Environnement virtuel pour Flask

- ✓ Ouvre le cmd de dos, crée un environnement virtuel et lance-le

```
python -m venv env  
env\Scripts\activate
```

- ✓ Vérifie si pip (outil d'installation pour python) est bien installé (l'aide de pip doit apparaître si c'est le cas)

```
(env) > pip help
```

- ✓ Dans le cmd de dos, installe flask

```
(env) > pip install flask
```

### 3. SQLite3

- ✓ Vas sous [www.sqlite3.org](http://www.sqlite3.org) dans la rubrique Download -> Precompiled Binaries for Windows et télécharge 64-bit DLL (x64) for SQLite version 3 (les 3 fichiers .exe sont aussi fournis dans le drive)
- ✓ Ouvre l'explorateur de fichier et décompresse le fichier zip. **Les 3 fichiers .exe devront systématiquement être placés dans ton dossier de travail.**
- ✓ Ouvre le cmd de dos dans ton dossier de travail :



- ✓ Tu peux lancer sqlite3 à partir du cmd mais vérifie toujours que tes 3 fichiers .exe sont bien placés dans le dossier d'où tu lances sqlite3

### 4. Visual Studio Code

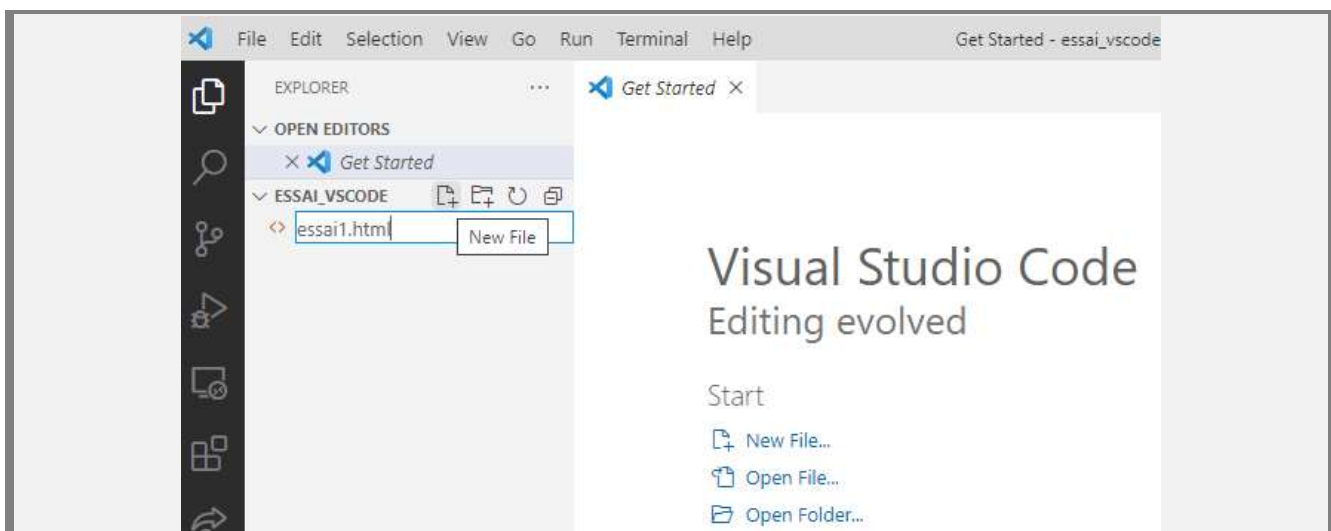
L'exécutable d'installation est disponibles sous :

<https://code.visualstudio.com/download>

## 2.4 Configuration de vs code

### 1. HTML

- ✓ Par l'explorateur de fichiers, crée un dossier vscode
- ✓ Ouvre vs code et ouvre ce dossier
- ✓ Avec le menu de gauche, crée un fichier essai1.html



- ✓ Ecris html et sélectionne **html:5** afin de faire apparaître le template html.
- ✓ Complète-le par « **<h1> hello en html !</p>** » et ouvre-le dans ton navigateur par l'explorateur de fichier

## 2. Python

- ✓ Crée un fichier `essai2.py`
- ✓ Quand tu click dans le code, accepte l'installation des extensions et écris :

```
print("Hello ...")  
print("... en python")  
for loop in range(5):  
    print("Comment vas-tu ?")  
print("T'es sûr que ça va ?")
```

- ✓ Exécute ton code et observe dans le terminal du bas:

**Run -> Run without debugging**


- ✓ Pointe avec la souris sur le numéro de la ligne 2 pour rajouter un point d'arrêt

-> clique droit

-> **add a breakpoint**

- ✓ Exécute ton code en mode debug

**run -> start debugging ou F5 ou **

- ✓ Le code va s'arrêter sur la ligne 2, joue avec les commandes  pour avancer pas à pas dans la boucle et visualiser le contenu de la variable sur la gauche.
- ✓ Pour ouvrir un terminal python (dans lequel tu peux taper des instructions python) :

**-> View -> Command palette -> python : Start REPL**

## 3. SQLite

- ✓ Recherche les extensions SQLite (menu de gauche) et installe la 1<sup>ère</sup> proposée :



- ✓ Crée un fichier `essai.db` dans lequel la bd sera sauvegardée et ouvre là :

clique droit -> open database

- ✓ Crée un fichier `essai.sql` et écris les requêtes suivantes :

```
DROP TABLE IF EXISTS bandedessinee;  
CREATE TABLE bandedessinee (id integer PRIMARY KEY, titre TEXT, auteur TEXT);
```

```
INSERT INTO bandedessinee VALUES(1, 'tintin au congo', 'hergé');  
select * from bandedessinée;
```

- ✓ Sélectionne le code et lance les requêtes :

clique droit -> run selected query

#### 4. Flask

Ouvre un cmd par l'explorateur de fichier et active l'environnement virtuel (voir plus haut)

Ferme puis re-ouvre visual studio code dans l'environnement virtuel.

(end) D : ... essai\_vscode > code .

- ✓ Recherche les extensions flask et installe la 1<sup>ère</sup> proposée.  
Profite-s-en pour découvrir les snippets proposés (notamment **hw**)
- ✓ Ouvre la palette de commande (**Open -> commande palette**), recherche python, sélectionne Python : **Select Interpreter** puis celui correspond à l'environnement virtuel
- ✓ Crée un fichier **essai.py** et récupère le template flask (en tapant **hw** dans le code)
- ✓ Dans le terminal, lance le code :

**python view.py**

- ✓ click sur le lien pour ouvrir le programme sur le serveur

### 3 Le web

Le "**World Wide Web**", plus communément appelé "Web" a été développé au CERN (Conseil Européen pour la Recherche Nucléaire) par le Britannique Sir Timothy John Berners-Lee et le Belge Robert Cailliau au début des années 90. À cette époque les principaux centres de recherche mondiaux étaient déjà connectés les uns aux autres, mais pour faciliter les échanges d'information Tim Berners-Lee met au point le **système hypertexte**. Le système hypertexte permet, à partir d'un document, de consulter d'autres documents en cliquant sur des mots clés. Ces mots "cliquables" sont appelés hyperliens et sont souvent soulignés et en bleu. Ces hyperliens sont plutôt connus aujourd'hui sous le simple terme de "**liens**".

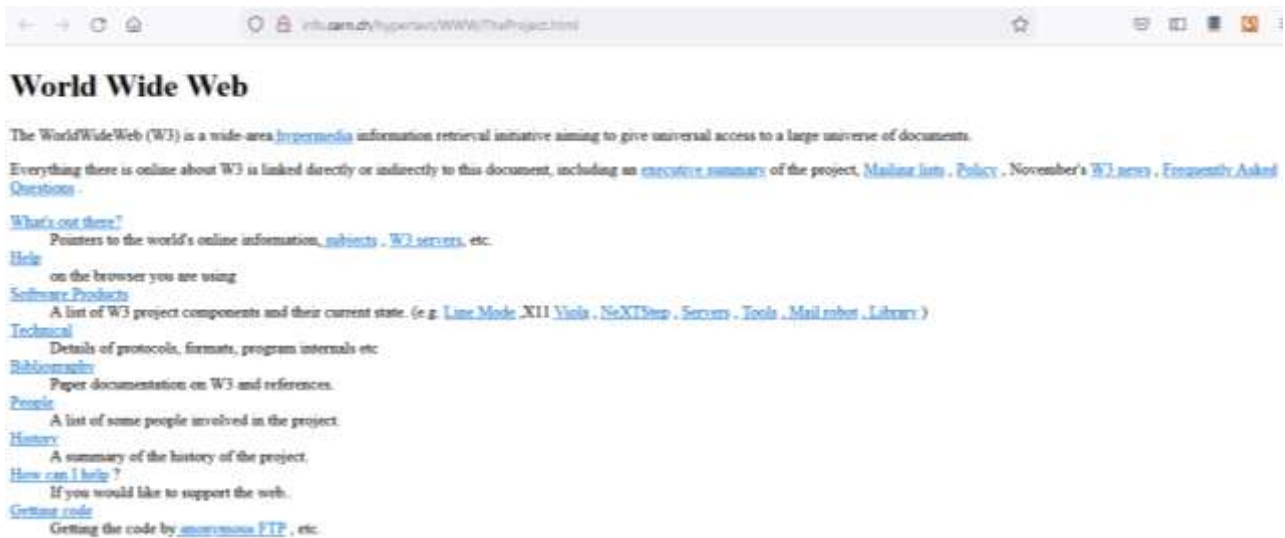


Figure 1: première page web, les hyperliens sont soulignés et en bleu. Cette première page web est toujours consultable à l'adresse suivante : <http://info.cern.ch/hypertext/WWW/TheProject.html>

Tim Berners-Lee développe le premier **navigateur web** (**logiciel permettant de lire des pages contenant des hypertextes**), il l'appelle simplement "**WorldWideWeb**".

Il faudra attendre 1993 et l'arrivée du navigateur web "NCSA Mosaic" pour que le web commence à devenir populaire en dehors du petit monde de la recherche.

Techniquement le web se base sur trois choses :

- ✓ le **protocole HTTP** (HyperText Transfert Protocol),
- ✓ les **URL** (Uniform Resource Locator)
- ✓ le **langage de description HTML** (HyperText Markup Language).

Nous aurons, très prochainement l'occasion de revenir sur ces trois éléments.

Une chose très importante à bien avoir à l'esprit : beaucoup de personnes confondent "web" et "internet". Même si le "web" "s'appuie" sur internet, les deux choses n'ont rien à voir puisqu'"**internet**" est un "**réseau de réseau**" s'appuyant sur le **protocole IP** alors que, comme nous venons de le voir, le **web** est la combinaison de trois technologies : **HTTP, URL et HTML**. D'ailleurs on trouve autre chose que le "web" sur internet, par exemple, les emails avec le protocole SMTP (Simple Mail Transfert Protocol) et les transferts de fichiers avec le protocole FTP (File Transfert Protocol).



## 4 Les URL

Dans la barre d'adresse de votre navigateur web vous trouvez, quand vous visitez un site, des choses du genre :

[https://qkzk.xyz/docs/nsi/cours\\_premiere/programmation/comparaison/](https://qkzk.xyz/docs/nsi/cours_premiere/programmation/comparaison/)

La partie `"/docs/nsi/cours_premiere/programmation/comparaison/"` s'appelle une **URL**. Plus précisément :

protocole	domaine	URL
https://	qkzk.xyz	/docs/nsi/cours_premiere/programmation/comparaison/

Une **URL** (**Uniform Resource Locator**) permet d'identifier une ressource (par exemple un fichier) sur un réseau.

L'URL indique « l'endroit » où se trouve une ressource sur un ordinateur. Un fichier peut se trouver dans un dossier qui peut lui-même se trouver dans un autre dossier. . . On parle d'une structure en **arborescence**, car elle ressemble à un arbre à l'envers :

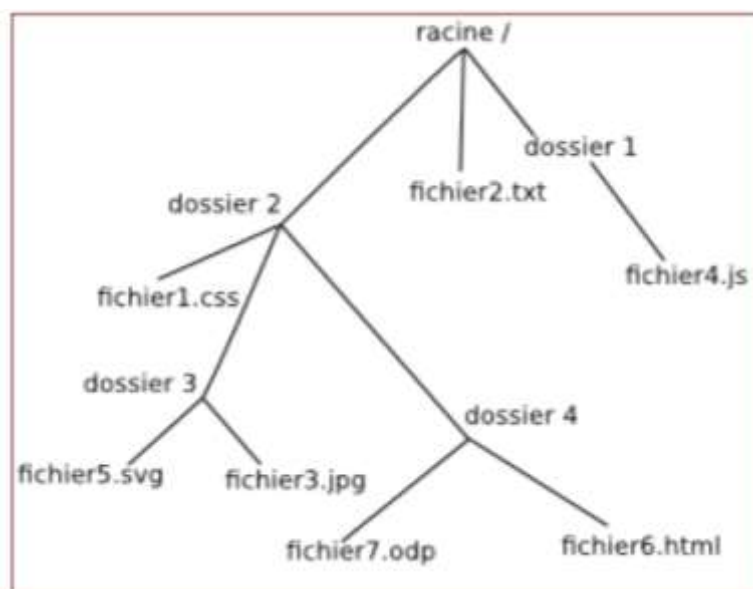


Figure 2: structure en arborescence

Comme vous pouvez le constater, la base de l'arbre s'appelle la racine de l'arborescence et se représente par un /

### Chemin absolu ou chemin relatif ?

Pour indiquer la position d'un fichier (ou d'un dossier) dans l'arborescence, il existe 2 méthodes :

- ✓ indiquer un chemin absolu
- ✓ indiquer un chemin relatif.

Le **chemin absolu** doit indiquer « le chemin » depuis la **racine**.

Par exemple l'URL du fichier fichier3.jpg sera : /dossier2/dossier3/fichier3.jpg

Remarquez que nous démarrons bien de la racine / (attention les symboles de séparation sont aussi des /)



Imaginons maintenant que le fichier fichier1.css fasse appel au fichier fichier3.jpg (comme un fichier HTML peut faire appel à un fichier CSS). Il est possible d'indiquer le chemin non pas depuis la racine, mais depuis le dossier (dossier2) qui accueille le fichier1.css, nous parlerons alors de **chemin relatif** :

dossier3/fichier3.jpg

Remarquez l'absence du / au début du chemin (c'est cela qui nous permettra de distinguer un chemin relatif et un chemin absolu).

Imaginons maintenant que nous désirions indiquer le chemin relatif du fichier fichier1.css depuis l'intérieur du dossier dossier4.

Comment faire ?

Il faut « remonter » d'un « niveau » dans l'arborescence pour se retrouver dans le dossier dossier2 et ainsi pouvoir repartir vers la bonne « branche ».

Pour ce faire il faut utiliser 2 points :..

../dossier2/fichier1.css

Il est tout à fait possible de remonter de plusieurs « crans » : ../../ depuis le dossier dossier4 permet de « retourner » à la racine.

### En résumé :

- ✓ **Chemin absolu**
  - débute par /
  - indique l'adresse depuis la racine
- ✓ **Chemin relatif**
  - ne débute par / (ou alors par ./)
  - indique l'adresse depuis la position actuelle
  - ../ signifie un niveau plus haut dans l'arborescence

### Exercice

Soit la structure en arborescence suivante:

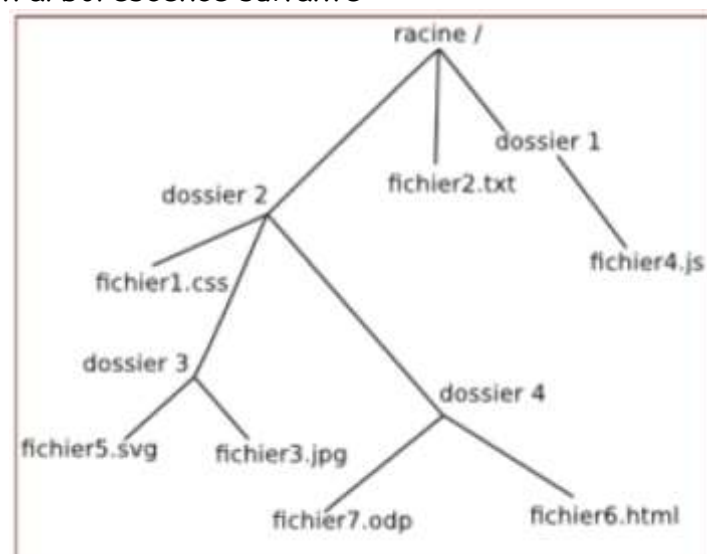


Figure 3: arborescence

1. Le contenu du fichier "fichier7.odp" utilise le fichier "fichier5.svg". Donnez le chemin relatif qui devra être renseigné dans le fichier "fichier7.odp" afin d'atteindre le fichier "fichier5.svg".
2. Donnez le chemin absolu permettant d'atteindre le fichier "fichier6.html".

Remarque : la façon d'écrire les chemins (avec des slash (/) comme séparateurs) est propre aux systèmes dits « UNIX », par exemple GNU/Linux ou encore Mac OS. Sous Windows, ce n'est pas le slash qui est utilisé, mais l'antislash (\). Pour ce qui nous concerne ici, les chemins réseau (et donc le web), pas de problème, c'est le slash qui est utilisé.

## 5 HTML et CSS : rappels

Le couple HTML + CSS sont les deux langages utilisés pour présenter et mettre en forme les documents sur le web.

### 5.1 HTML

- ✓ HTML est un **langage de balise** et non un langage de programmation.
- ✓ HTML permet de décrire le **contenu** d'un document et d'en donner la **structure**.
- ✓ HTML n'étant pas un langage de programmation, il n'existe pas de variables ou de conditions, fonctions etc en HTML.

Voici la définition accessible sur Wikipédia :

*L'Hypertext Markup Language, généralement abrégé HTML, est le format de données conçu pour représenter les pages web. C'est un langage de balisage permettant d'écrire de l'hypertexte, d'où son nom. HTML permet également de structurer sémantiquement et de mettre en forme le contenu des pages, d'inclure des ressources multimédias, dont des images, des formulaires de saisie, et des programmes informatiques. Il permet de créer des documents interopérables avec des équipements très variés de manière conforme aux exigences de l'accessibilité du web. Il est souvent utilisé conjointement avec des langages de programmation (JavaScript) et des formats de présentation (feuilles de style en cascade).*

#### Langage de balise

- ✓ Le principe des balises est **d'encapsuler un contenu** entre une balise ouvrante et une balise fermante.

#### Exemple

```
<div>
  Le contenu de cette balise div
</div>
```

- ✓ L'action de ces balises porte alors sur la partie du texte qu'elle contient.
- ✓ Les balises peuvent être imbriquées.
- ✓ De la même manière qu'une série de parenthèse comme ceci : [( )] ne convient pas, les balises doivent être correctement ordonnées :

```
<div>
  <p>
    Un paragraphe
  </p>
</div>
</p>
```

Cet exemple ne convient pas ! Comment le rectifier ?

## Structure des pages web

- ✓ L'ordre dans lequel sont décrites les balises permet de construire une structure de la page.
- ✓ Tout en haut, à la racine de cet arbre, on trouve la balise `<html>`.
- ✓ L'ensemble du document est compris entre `<html>` et `</html>`
- ✓ Dans ce contenu on trouve alors deux parties :
  - `<head>` qui contient la description de **l'entête** (nom de l'onglet, propriétés parfois les fichiers importés etc.)
  - `<body>` qui contient le **corps de la page**.

```
<html>
  <head>
    La description du document
  </head>
  <body>
    Ce qui sera affiché dans la page
  </body>
</html>
```

Intéressons-nous au document lui-même.

Les balises qu'il contient ont toutes un rôle :

- `<div>` décrire un bloc qui sera aligné verticalement avec les autres blocs ;
- `<span>` décrire un bloc qui sera aligné horizontalement avec les autres blocs ;
- `<p>` décrire un paragraphe ;
- `<h1>`, `<h2>` etc. décrire un titre, un sous-titre etc. ;
- `<form>` contient un formulaire permettant à l'utilisateur d'envoyer des informations au serveur ;
- `<button>` un bouton (qui l'eut cru ?) ;
- `<ol>` `<ul>` listes numérotées ou non ;
- `<li>` un élément d'une liste
- `<table>` un tableau... dont la structure est aussi séparée entre entête `<thead>` et contenu `<tbody>`
- `<tr>` table row, une ligne et `<td>` table data, une cellule de la ligne.
- `<a>` un lien ;
- `<img>` une image ... etc.

Remarquons que ces balises visent toutes à décrire la **structure de la page** et non sa mise en forme.

Deux constats immédiats :

1. La **mise en forme** du document doit pouvoir être faite de manière automatique, si possible en décrivant ce qu'on souhaite faire. C'est le rôle de **CSS**
2. La construction de cette structure respecte une organisation simple. Dans la majorité des cas, il est possible de la générer automatiquement ! C'est en partie les rôles de python ou javascript que nous aborderons plus tard.

## D'une structure à son arbre

```
<html>
  <head>
    <title>Le titre dans l'onglet du navigateur</title>
  </head>
  <body>
    <div>
      <h1>Un titre</h1>
      <p>Un paragraphe</p>
    </div>
    <div>
      <table>
        <caption>Alien football stars</caption>
        <tr>
          <th scope="col">Player</th>
          <th scope="col">Gloobles</th>
          <th scope="col">Za'taak</th>
        </tr>
        <tr>
          <th scope="row">TR-7</th>
          <td>7</td>
          <td>4,569</td>
        </tr>
        <tr>
          <th scope="row">Khiroesh Odo</th>
          <td>7</td>
          <td>7,223</td>
        </tr>
        <tr>
          <th scope="row">Mia Oolong</th>
          <td>9</td>
          <td>6,219</td>
        </tr>
      </table>
    </div>
    <div>
      <ul>
        <li>Milk</li>
        <li>Cheese
          <ul>
            <li>Blue cheese</li>
            <li>Feta</li>
          </ul>
        </li>
      </ul>
    </div>
  </body>
</html>
```

Construire l'arbre (appelé **DOM**) décrivant la structure de cette page. Attention... il est imposant !

Pour vérifier : <https://yoksel.github.io/html-tree/en>

## Propriétés d'une balise

Il est possible d'inclure des propriétés DANS la description d'une balise. Ces propriétés s'appliqueront à tout le texte sélectionné par cette balise. Par exemple :

```
<div class="ma-classe">
  Une div avec la classe "ma-classe"
  Plusieurs balises peuvent porter la même classe.
</div>
<p id="mon-id">
  Un paragraphe avec l'id "mon-id"
  Une seule balise par id.
</p>
<p style="color : blue">
  Un paragraphe avec la propriété « color » bleue
</p>
```

## 5.2 CSS

Si HTML décrit la **structure** du document, CSS en réalise la **mise en forme**.

C'est CSS qui permet d'appliquer des couleurs, des changements de fontes, de taille etc.

CSS pour "**Cascading Style Sheets**" ou "feuilles de style en cascade"

Le principe de CSS est d'appliquer une mise en forme à un ou des éléments sélectionnés.

Ainsi :

```
p {  
  color: red;  
}
```

Va :

1. sélectionner tous les paragraphes contenus dans des balises <p>
  2. leur appliquer les commandes entre les { }
- Dans notre exemple, color désigne la couleur du texte, qui sera rouge.

### Anatomie d'une règle CSS

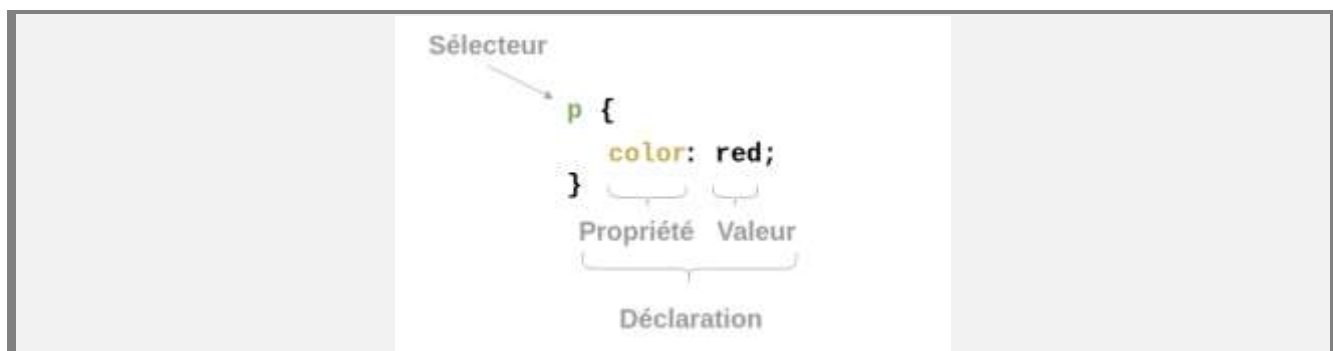


Figure 1: règle css

#### ✓ Sélecteur

C'est le nom de l'élément HTML situé au début de l'ensemble de règles. Il permet de sélectionner les éléments sur lesquels appliquer le style souhaité (en l'occurrence, les éléments p). Pour mettre en forme un élément différent, il suffit de changer le sélecteur.

#### ✓ Déclaration

C'est une règle simple comme 'color: red;' qui détermine les propriétés de l'élément que l'on veut mettre en forme.

#### ✓ Propriétés

Les différentes façons dont on peut mettre en forme un élément HTML (dans ce cas, 'color' est une propriété des éléments 'p'). En CSS, vous choisissez les différentes propriétés que vous voulez utiliser dans une règle CSS.

## Valeur de la propriété

À droite de la propriété, après les deux points, on a la valeur de la propriété. Celle-ci permet de choisir une mise en forme parmi d'autres pour une propriété donnée (par exemple, il y a d'autres couleurs que red pour la propriété color).

Les autres éléments importants de la syntaxe sont :

- chaque ensemble de règles, à l'exception du sélecteur, doit être entre accolades ({}).
- pour chaque déclaration, il faut utiliser deux points (:) pour séparer la propriété de ses valeurs.
- pour chaque ensemble de règles, il faut utiliser un point-virgule (;) pour séparer les déclarations entre elles.

Ainsi, si on veut modifier plusieurs propriétés d'un coup, on peut utiliser plusieurs déclarations dans une seule règle en les séparant par des points-virgules :

```
p {  
  color: red;  
  width: 500px;  
  border: 1px solid black;  
}
```

## Sélectionner plusieurs éléments

Il est aussi possible de sélectionner plusieurs types d'éléments pour appliquer à tous une même règle. Il suffit de placer plusieurs sélecteurs, séparés par des virgules.

Par exemple :

```
p, li, h1 {  
  color: red;  
}
```

## Les différents types de sélecteurs

Il y a différents types de sélecteurs. Dans les exemples précédents, nous n'avons vu que les sélecteurs d'élément qui permettent de sélectionner les éléments HTML d'un type donné dans un document HTML. Mais ce n'est pas tout, il est possible de faire des sélections plus spécifiques.

Voici quelques-uns des types de sélecteur les plus fréquents :

### ✓ Les sélecteurs de balise.

Ils sélectionnent toutes les balises portant ce nom.

```
p {  
  color: red;  
}
```



### ✓ Les sélecteurs d'ID

Ils sélectionnent sélectionne LA balise portant cette id (on ne peut avoir qu'un seul élément pour un ID donné)

#### Code css

```
#mon-id {  
    background-color: black;  
}
```

#### Code html

```
<a id="mon-id">
```

### ✓ Les sélecteurs de classes

Ils sélectionnent TOUTES les balises portant cette classe

#### Code css

```
.ma-classe {  
    font-style: verdana;  
}
```

#### Code html

```
<a class="ma-classe">
```

### ✓ Les sélecteurs de "pseudo-classe"

Ils sélectionnent les éléments donnés mais uniquement dans un certain état

```
a:hover {  
    color: blue;  
}
```

'a:hover' sélectionne les liens uniquement quand la souris passe dessus

## Les grands types de mise en forme

On peut séparer en quelques grandes catégories les propriétés appliquées :

### ✓ Les polices de caractères

Ce sont les propriétés qu'on applique au texte lui-même : couleur, taille, épaisseur, fonte etc.

### ✓ Les boîtes

Ces propriétés décrivent les marges intérieures (padding) et extérieures (margin) ainsi que la bordure (border) et la taille en largeur, hauteur, la couleur de fond etc.

### ✓ L'alignement

Souhaite-t-on aligner horizontalement, verticalement, avoir une image positionnée précisément ou à une position dépendant des dimensions de la page ?

## La fenêtre de développement



Figure 4: html et css

C'est devenu depuis une dizaine d'année l'outil de base du développeur web. Tous les navigateurs modernes permettent de consulter le code d'une page et de l'éditer localement. On peut ainsi appliquer le style, déboguer les erreurs et mesurer les performances dans un outil complet et intégré au navigateur.

### Commande pour accéder à l'éditeur de code firefox

Menu en haut à gauche -> outils supplémentaires -> outils de développement

ou le raccourci

Ctrl + Maj + I

### 5.3 Test1 : rappels sur les liens html (<a> ... </a>)

Sous **public\_html**, crée un dossier **uaa12\_web**, puis un sous dossier **0\_tp\_intro**  
Dans ce dernier dossier, crée 2 fichiers html **test1p1.html** et **test1p2.html** à partir du **template html** et code les liens pour obtenir le résultat ci-dessous.  
Déplace ensuite test1p2.html dans un dossier **test1p2** et mets à jour ton code pour que les liens soient fonctionnels.

#### test1p1.html dans le navigateur

Bienvenue sur le page 1

Clique sur [ce lien](#) pour accéder à la page 2

#### test1p2.html dans le navigateur

Bienvenue sur le page 2

Clique sur [ce lien](#) pour revenir à la page 1

## 6 Modèle Client - Serveur

### 6.1 Qu'est-ce que le modèle client-serveur ?

Deux ordinateurs en réseau peuvent s'échanger des données. Dans la plupart des cas ces échanges ne sont pas "symétriques" : en effet un ordinateur A va souvent se contenter de demander des ressources (fichiers contenant du texte, photos, vidéos, sons...) à un ordinateur B. L'ordinateur B va lui se contenter de fournir des ressources à tous les ordinateurs qui lui en feront la demande. On dira alors que l'ordinateur A (**celui qui demande des ressources**) est un **client** alors que l'ordinateur B (**celui qui fournit les ressources**) sera qualifié de **serveur**.

En tapant «<http://www.google.fr>», votre machine va chercher à entrer en communication avec le serveur portant le nom «[www.google.fr](http://www.google.fr)».

Une fois la liaison établie, le client et le serveur vont échanger des informations en dialoguant :

- client : bonjour [www.google.fr](http://www.google.fr), pourrais-tu m'envoyer le fichier index.html ?
- serveur : OK client, voici le fichier index.html
- client : je constate que des images, du code css sont utilisés, peux-tu me les envoyer ?
- serveur : OK, les voici

Évidemment ce dialogue est très imagé, mais il porte tout de même une part de « vérité ».

Sur internet, ce modèle client/serveur domine assez largement, même s'il existe des cas où un ordinateur pourra jouer tour à tour le rôle de client et le rôle de serveur, très souvent, des ordinateurs (les clients) passeront leur temps à demander des ressources à d'autres ordinateurs (les serveurs) . Par exemple, comme expliqué dans l'exemple ci-dessus on retrouve cet échange client/serveur à chaque fois que l'on visite une page web. Il y a de fortes chances pour que votre ordinateur personnel joue quasi exclusivement le rôle de client (sauf si vous êtes un adepte du "peer to peer").

N'importe quel type d'ordinateur peut jouer le rôle de serveur, mais dans le monde professionnel les serveurs sont des machines spécialisées conçues pour fonctionner 24h sur 24h. Ils peuvent aussi avoir une grosse capacité de stockage afin de stocker un grand nombre de ressources (vidéos, sons,...). Une seule machine peut servir de nombreuses applications.

#### Un serveur

Afin assurer une continuité de service, dans les sociétés, plusieurs serveurs assurent exactement le même rôle (on parle de redondance). Vous vous doutez bien que Google ne possède pas qu'un seul serveur, en effet, en moyenne, chaque seconde, c'est environ 65000 clients qui se connectent aux serveurs du moteur de recherche de Google.

Aucun serveur, même extrêmement performant, ne serait capable de répondre à toutes ces requêtes. Google, Amazon, Facebook ou Netflix possèdent un très grand nombre de serveurs afin de pouvoir satisfaire les demandes des utilisateurs en permanence. Ces entreprises possèdent d'immenses salles contenant chacune des

centaines ou des milliers de serveurs (ces serveurs sont rangés dans des armoires appelées "baie serveur").



Figure 5 : serveur

## Salle serveur

Souvent les serveurs sont spécialisés dans certaines tâches, par exemple, les serveurs qui envoient aux clients des pages au format HTML sont appelés "**serveur web**".



Figure 6 : salle serveur

## 6.2 Sites statiques

Langages Il y a quelques années, le web était dit « **statique** » : le concepteur de site web écrivait son code HTML et ce code était simplement envoyé par le serveur web au client. Les personnes qui consultaient le site avaient toutes le droit à la même page, le web était purement « **consultatif** ».



Figure 7 : site statique

La communication est donc plutôt basique :

- « Bonjour, je suis le client, je voudrais voir cette page web. »
- « Tiens, voilà la page que tu m'as demandée. »

## 6.3 Sites dynamiques

Les choses ont ensuite évolué : les serveurs sont aujourd'hui **capables de générer eux-mêmes du code HTML**. Les résultats qui s'afficheront à l'écran dépendront donc des demandes effectuées par l'utilisateur du site : le web est devenu **dynamique**.

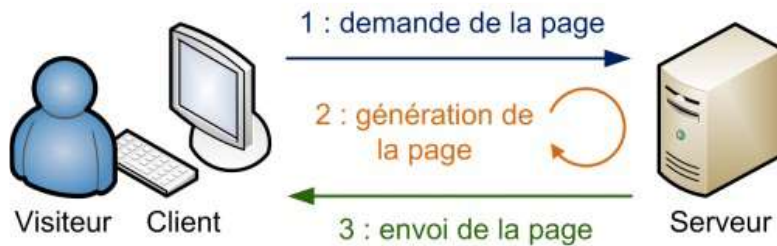


Figure 8 : site dynamique

Il y a une étape intermédiaire : la page est générée.

- le client demande au serveur à voir une page web ;
- le serveur prépare la page spécialement pour le client ;
- le serveur lui envoie la page qu'il vient de générer.

## 6.4 Langages de programmation côté serveur

Différents **langages de programmation** peuvent être utilisés « **côté serveur** » afin de permettre au serveur de générer lui-même le code HTML à envoyer. Les plus utilisés aujourd'hui pour générer des pages web côté serveur sont **PHP, Java et Python**. On trouve aussi Javascript avec node.js

PHP par exemple génère du code html, code qui est ensuite envoyé au client de la même manière qu'un site statique.



Figure 9 : page web générée par le serveur

Voici un exemple très simple de code en PHP :

### Code de `essai2.php`

```
<?php
$heure = date("H:i");
echo '<h1>Bienvenue sur mon site</h1>' ;
echo '<p>Il est '.$heure.'</p>';
?>
```

Et voici une fonction en python flask qui générera le même résultat :

## Code de view.py

```
def bienvenue():  
    tactuel=localtime(time())  
    heure="<h1>Bienvenue sur mon site</h1>"+str(tactuel.tm_hour)+" h "+str(tactuel.tm_min)+" mn "  
    return heure
```

Sans entrer dans les détails, si un client se connecte à un serveur web qui exécute ce code à 18h23, le serveur enverra au client le code HTML ci-dessous :

```
<h1>Bienvenue sur mon site</h1>  
<p>Il est 18h23</p>
```

En revanche si un client se connecte à ce même serveur à 9h12, le serveur enverra au client le code HTML ci-dessous :

```
<h1>Bienvenue sur mon site</h1>  
<p>Il est 9h12</p>
```

Comme vous pouvez le constater, le PHP permet de générer des pages HTML dynamiquement. Inutile de préciser que cet exemple est volontairement très simple, le PHP est capable de générer des pages HTML bien plus complexes. En particulier l'exemple ci-dessus est parfaitement réalisable en Javascript côté client (voir paragraphe suivant). Ce que le Javascript côté client ne permet pas de faire est **d'écrire des informations sur le serveur**. Ainsi, pour créer des **comptes utilisateurs**, **accéder à une base de données**, il est indispensable d'avoir des instructions côté serveur.

Nous allons apprendre à générer dynamiquement des pages web côté serveur en utilisant **Python Flask** très prochainement.

## 6.5 Langages de programmation côté client

Tous les langages modernes peuvent être utilisés pour générer une page web automatiquement et structurer des données. Par contre, alors que php ou python sont exécuté côté **serveur**, **Javascript** est le seul qui puisse être exécuté dans le **navigateur du client**.

En quoi est-ce intéressant ? Cela permet de réaliser les calculs sur la machine du client et non sur le serveur directement. Aussi on **limite la charge du serveur** !

La finalité des scripts côté client et côté serveur n'est pas la même : un script côté serveur va s'occuper de créer la page Web qui sera envoyée au navigateur. Ce dernier va alors afficher la page puis exécuter les scripts côté client tel que le Javascript.



Figure 10 : page web interprétée par le client

Nous allons apprendre à générer dynamiquement des pages web côté client en utilisant **Javascript** très prochainement.

Voici le code javascript qui générera le même message de bienvenue qu'en php :

#### Code de hello\_js.html

```
<script>
  var now = new Date();
  var heure  = now.getHours();
  var minute = now.getMinutes();
  var seconde = now.getSeconds();
  alert('Il est ' + heure + 'h ' + minute + 'min ' + seconde + 's');
</script>
```



## 6.6 Test2 - Accéder à un pc du local via le serveur Apache

1. Télécharge du drive tout le dossier UAA12\_Web, extrais-le et copie-le sous public.html.
2. Dans l'URL (barre d'adresse) du navigateur internet , connecte-toi sur le serveur de ton PC en tapant « **localhost** ».

La page « Apache2 Debian Default Page » doit s'afficher :



3. Accède à ton dossier utilisateur (nom avec lequel tu t'es logué sur ton PC) en tapant dans l'URL (barre d'adresse) de ton navigateur:

**localhost/~utilisateur**

Tu as alors accès aux dossiers et fichiers stockés sous :

**/home/utilisateur/public\_html**

4. Descend dans l'arborescence et double click sur les fichiers suivants :
  - essai1.html (écrit en html)
  - hello\_php.php (écrit en php, fichier téléchargé du drive)
  - hello\_js.html (écrit en javascript, fichier téléchargé du drive)
5. Edite les fichiers avec gedit et compare le code html généré par Apache sur chacun de ces 2 fichiers (dans le navigateur -> souris sur la page -> click droit -> code source de la page).
6. Ouvre un terminal, et lis l'adresse IP de ton pc.

**hostname -I**

7. Donne-là à ton voisin pour qu'il accède à tes fichiers dans ton dossier utilisateur en tapant dans l'URL de son navigateur :

**adresse\_ip /~utilisateur**

## 7 Protocole HTTP

### 7.1 Requête HTTP

Revenons sur l'adresse qui s'affiche dans la barre d'adresse d'un navigateur web et plus précisément sur le début de cette adresse c'est-à-dire le "http"

Selon les cas cette adresse commencera par http ou https (« s » pour sécurisé).

Le protocole (un **protocole** est **ensemble de règles qui permettent à 2 ordinateurs de communiquer ensemble**) HTTP (**HyperText Transfert Protocol**) va permettre au client d'effectuer des **requêtes** à destination d'un serveur web. En retour, le serveur web va envoyer une **réponse**.

Voici une version simplifiée de la composition d'une requête HTTP (client vers serveur) :

- la méthode employée pour effectuer la requête
- l'URL de la ressource
- la version du protocole utilisé par le client (souvent HTTP 1.1)
- le navigateur employé (Firefox, Chrome) et sa version
- le type du document demandé (par exemple HTML) ...

Certaines de ces lignes sont optionnelles.

Voici un exemple de requête HTTP :

```
GET /mondossier/monFichier.html HTTP/1.1
```

```
User-Agent : Mozilla/5.0
```

```
Accept : text/html
```

Nous avons ici plusieurs informations :

- "GET" est la **méthode employée** (voir ci-dessous)
- "/mondossier/monFichier.html" correspond à **l'URL de la ressource demandée**
- "HTTP/1.1" : **la version du protocole est la 1.1**
- "Mozilla/5.0" : le **navigateur web employé est Firefox de la société Mozilla**
- "text/html" : le client s'attend à recevoir du **HTML**

Revenons sur la méthode employée :

Une requête HTTP utilise une **méthode** (c'est une **commande qui demande au serveur d'effectuer une certaine action**).

Voici la liste des méthodes disponibles :

```
GET, HEAD, POST, OPTIONS, CONNECT, TRACE, PUT, PATCH, DELETE
```

Détaillons 4 de ces méthodes :

- **GET** : C'est la méthode la plus courante pour demander une ressource. Elle est sans effet sur la ressource.
- **POST** : Cette méthode est utilisée pour soumettre des données en vue d'un traitement (côté serveur). Typiquement c'est la méthode employée lorsque l'on envoie au serveur les données issues d'un **formulaire**.
- **DELETE** : Cette méthode permet de *supprimer* une ressource sur le serveur.
- **PUT** : Cette méthode permet de modifier une ressource sur le serveur

## 7.2 Réponse du serveur à une requête HTTP

Une fois la requête reçue, le serveur va renvoyer une réponse, voici un exemple de réponse du serveur :

```
HTTP/1.1 200 OK
Date: Thu, 15 feb 2019 12:02:32 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Voici mon site</title>
</head>
<body>
  <h1>Hello World! Ceci est un titre</h1>
  <p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
</body>
</html>
```

Nous n'allons pas détailler cette réponse, voici quelques explications sur les éléments qui nous seront indispensables par la suite :

Commençons par la fin (à partir de `<!doctype html>`): **le serveur renvoie du code HTML**, une fois ce code reçu par le client, il est interprété par le navigateur qui affiche le résultat à l'écran. Cette partie correspond au corps de la réponse.

La 1<sup>re</sup> ligne se nomme la **ligne de statut** :

- HTTP/1.1 : version de HTTP utilisé par le serveur
- 200 : code indiquant que le document recherché par le client a bien été trouvé par le serveur. Il existe d'autres codes dont un que vous connaissez peut-être déjà : le code 404 (qui signifie «Le document recherché n'a pu être trouvé»).

Les 5 lignes suivantes constituent l'**en-tête** de la réponse, une ligne nous intéresse plus particulièrement :

Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4

Le serveur web qui a fourni la réponse http ci-dessus a comme **système d'exploitation** une distribution GNU/Linux nommée "Debian" (pour en savoir plus sur GNU/Linux, c'est celle que nous utiliserons d'ailleurs). "Apache" est le coeur du serveur web puisque c'est ce **logiciel qui va gérer les requêtes http** (recevoir les requêtes http en provenance des clients et renvoyer les réponses http).

Il existe d'autres logiciels capables de gérer les requêtes http (nginx, microsoft-iis...). NGINX (engine x) dépasse depuis peu (juin 2019) Apache. NGINX et Apache sont installés sur 85% des serveurs web mondiaux.

### 7.3 Les étapes du protocole HTTPS

Le "HTTPS" est la version "**sécurisée**" du protocole HTTP. Par "sécurisé" on entend que les **données sont chiffrées** avant d'être transmises sur le réseau et que leur provenance est certifiée.

Voici les différentes étapes d'une communication client - serveur utilisant le protocole HTTPS :

- le client demande au serveur une connexion sécurisée (en utilisant "https" à la place de "http" dans la barre d'adresse du navigateur web)
- le serveur répond au client qu'il est OK pour l'établissement d'une connexion sécurisée. Afin de prouver au client qu'il est bien celui qu'il prétend être, le serveur fournit au client un certificat prouvant son "identité".

En effet, il existe des attaques dites "man in the middle", où un serveur "pirate" essaye de se faire passer, par exemple, pour le serveur d'une banque : le client, pensant être en communication avec le serveur de sa banque, va saisir son identifiant et son mot de passe, identifiant et mot de passe qui seront récupérés par le serveur pirate. Afin d'éviter ce genre d'attaque, des organismes délivrent donc des certificats prouvant l'identité des sites qui proposent des connexions "https".

- à partir de ce moment-là, les échanges entre le client et le serveur seront chiffrés grâce à un système de "clé publique - clé privée".

Même si un pirate arrivait à intercepter les données circulant entre le client et le serveur, ces dernières ne lui seraient d'aucune utilité, car totalement incompréhensibles à cause du chiffrement (seuls le client et le serveur sont aptes à déchiffrer ces données)

D'un point de vue strictement pratique il est nécessaire de bien vérifier que le protocole est bien utilisé (l'adresse commence par "https") avant de transmettre des données sensibles (coordonnées bancaires...). Si ce n'est pas le cas, passez votre chemin, car toute personne qui interceptera les paquets de données sera en mesure de lire vos données sensibles.

## 7.4 Test3 : établir une connexion HTTP directement dans Python, sans utiliser de navigateur

Il est parfaitement possible d'utiliser Python (ou n'importe quel langage moderne) pour établir une connexion avec un serveur web.

### 1. Importe la librairie qui gère les connexions http

Code dans le terminal python

```
>>> import requests
```

### 2. Etablis une connexion avec le site et affiche le résultat

```
>>> reponse = requests.get("https://qkzk.xyz")
>>> reponse
```

Résultat dans le terminal python

```
<Response [200]>
```

On peut voir que la connexion est bien établie correctement (200 signifie OK) entre le client (Python lui même) et le serveur (Github.com) qui héberge mon site.

### 3. Affiche le detail de la réponse du serveur

```
>>> reponse.headers
```

Résultat

```
{'Server': 'GitHub.com', 'Content-Type': 'text/html; charset=utf-8',
'Last-Modified': 'Wed, 09 Oct 2019 16:10:57 GMT', 'ETag': 'W/"5d9e0691-e152"',
'Access-Control-Allow-Origin': '*', 'Expires': 'Sat, 12 Oct 2019 09:20:09 GMT',
'Cache-Control': 'max-age=600', 'Content-Encoding': 'gzip',
# réponse tronquée
```

Le contenu est en html, comme on s'y attend.  
L'encodage est de l'utf-8.

### 4. Affiche le contenu de la réponse

```
>>> reponse.content
```

Résultat

```
b'<!DOCTYPE html>
<html lang="en">
<head>
<meta name="generator" content="Hugo 0.57.2" />
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>accueil| qkzk</title> ...'
```

```
# tronqué. La suite est le contenu complet de la page d'accueil du site...
```

```
</body></html>'
```

## 5. Affiche la requête transmise par le client lui-même au serveur

```
>>> reponse.request
```

### Résultat

```
<PreparedRequest [GET]>
```

### Code

```
>>> reponse.request.headers
```

### Résultat

```
{'User-Agent': 'python-requests/2.22.0',  
'Accept-Encoding': 'gzip, deflate',  
'Accept': '/*/*',  
'Connection': 'keep-alive'}
```

C'est la requête qui a été transmise quand on a tapé :

```
>>> reponse = requests.get("http://qkzk.xyz")
```

On a bien transmis une requête GET au serveur, avec toutes les informations voulues. C'est bien Python qui a transmis cette requête.

## 7.5 Requête GET et POST

En théorie, une requête **GET** permet **d'accéder à un contenu, sans le modifier**, et une requête **POST** **d'envoyer un nouveau contenu**.

Ainsi, presque toutes les requêtes effectuées sur internet sont des requêtes GET.

### Transmettre une information via une requête GET

Lorsqu'on transmet des informations via une requête GET, celles-ci sont transmises dans l'URL directement, sous la forme de paire "clé:valeur" :

<https://httpbin.org/get?cle1=valeur1&cle2=valeur2>

Ainsi, on a transmis deux informations : cle1: valeur1 et cle2: valeur2

Remarquez la syntaxe : ? après l'adresse, cle=valeur et & pour séparer les paires.

**Test3 : Fais une recherche via google sur le foot puis modifie là directement dans l'URL pour faire une recherche sur le rugby**

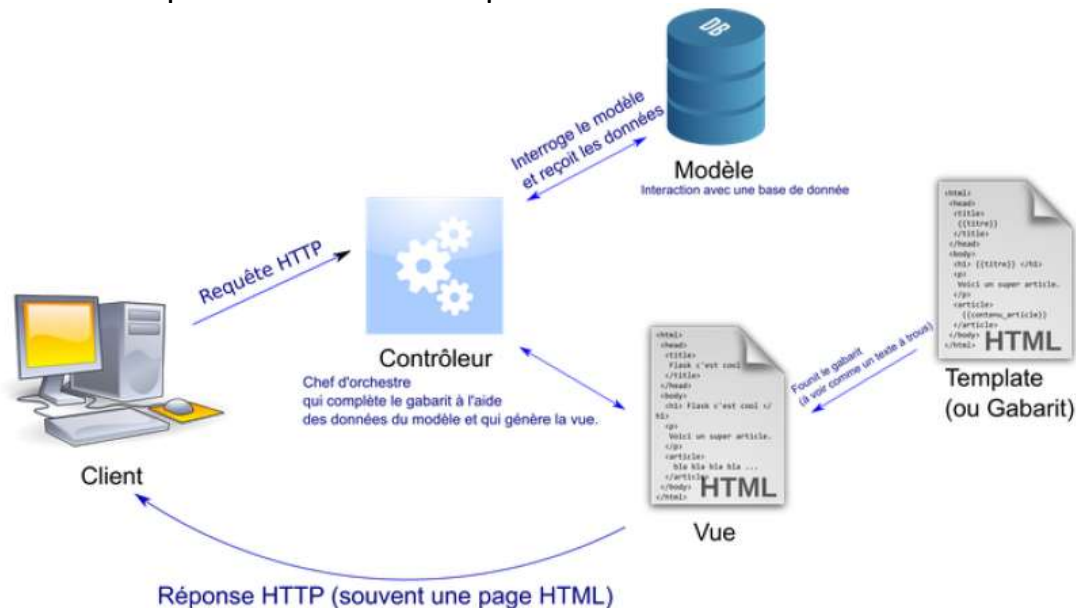
### Transmettre une information via une requête POST

Cette fois, on transmet l'information dans l'entête, généralement via le champ **form** (formulaire). Ainsi, lorsque vous vous identifiez ou remplissez un formulaire, vos informations sont généralement transmises via une requête POST.

Les informations ne sont pas visibles directement mais elles ne sont pas chiffrées pour autant. Simplement, le navigateur ne les affiche pas dans la page.

## 8 Le modèle MVC

L'architecture MVC est un modèle distinguant plusieurs rôles précis d'une application, qui doivent être accomplis. Comme son nom l'indique, l'architecture (ou « patron ») Modèle-Vue-Contrôleur est composée de trois entités distinctes, chacune ayant son propre rôle à remplir. Voici un schéma qui résume cela :



Le MVC permet de bien organiser son code source. Il va aider à savoir quels fichiers créer, mais surtout à définir leur rôle. Le but de MVC est justement de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- ✓ **Modèle** : cette partie **gère les données** de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les **requêtes aux bases de données**.
- ✓ **Vue** : cette partie se concentre sur **l'affichage**. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du **code HTML** mais aussi quelques boucles et conditions python très simples, pour afficher par exemple une liste de messages.
- ✓ **Contrôleur** : cette partie gère la **logique** du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. Le contrôleur contient exclusivement du **python**. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).
- ✓ **Template** : cette partie est le **modèle** de la page HTML qui sera utilisée par la vue pour générer la page HTML envoyée au client. On peut la voir comme un texte à trous dans lesquels seront insérées les données calculées par le contrôleur.



# Table des matières

1	Bibliographie et outils .....	1
2	Le web.....	7
3	Les URL.....	8
4	HTML et CSS : rappels.....	11
4.1	HTML .....	11
4.2	CSS .....	14
5	Modèle Client – Serveur .....	17
5.1	Qu'est-ce que le modèle client-serveur ?.....	18
5.2	Sites statiques .....	19
5.3	Sites dynamiques .....	20
5.4	Langages de programmation côté serveur .....	20
5.5	Langages de programmation côté client .....	21
5.6	Test1 – Accéder à un pc du local via le serveur Apache .....	23
6	Protocole HTTP .....	24
6.1	Requête HTTP .....	24
6.2	Réponse du serveur à une requête HTTP.....	25
6.3	Les étapes du protocole HTTPS .....	26
6.4	Test2 : établir une connexion HTTP directement dans Python, sans utiliser de navigateur.....	27
6.5	Requête GET et POST .....	28
7	Le modèle MVC.....	29