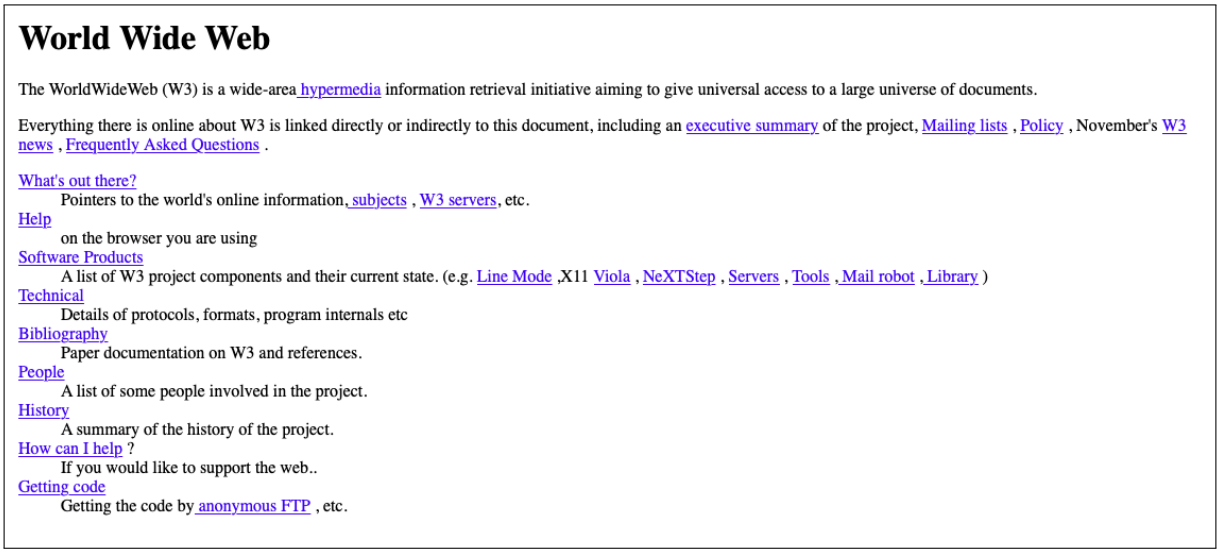
Le Web
d'après David Roche

1 Introduction

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".



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

FIGURE 1 – Première page web, les hyperliens sont soulignés 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) et 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 (voir le module Internet) 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).

2 Uniform Resource Locator

Dans la barre d'adresse de votre navigateur web vous trouverez, quand vous visitez un site, des choses du genre :
`http://www.gouv.mc/Gouvernement-et-Institutions/Le-Gouvernement/Ministere-d-Etat/Le-Ministre-d-Etat.html`

Nous aurons l'occasion de reparler du `http` et du `www.gouv.mc` plus tard. La partie `/Gouvernement- et-Institutions/Le-Go` s'appelle une URL.

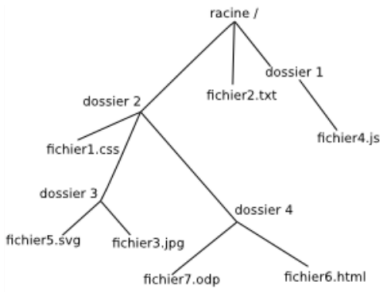


FIGURE 2 – Structure en arborescence

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 :

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 ou 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 `fichier5.svg` 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" (vers le `dossier3`). Pour ce faire il faut utiliser 2 points : `..`

```
../dossier3/fichier5.svg
```

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

Exercice 1 :

On considère la même structure d'arbre.

Le contenu du fichier `fichier7.odp` utilise le fichier `fichier5.svg`. Donnez le chemin relatif qui devra être renseigner dans le fichier `fichier7.odp` afin d'atteindre le fichier `fichier5.svg`.

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 MacOS. 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.

3 Introduction au l'HTML et au CSS

Nous allons nous intéresser à un acteur fondamental du développement web, le couple **HTML+CSS** (**Hyper Text Markup Langage** et **Cascading Style Sheets**).

Dans un premier temps, nous allons exclusivement nous intéresser au **HTML**. Qu'est-ce que le HTML, voici la définition que nous en donne Wikipedia :

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).

Pour l'instant, nous allons retenir deux éléments de cette définition « conçu pour représenter les pages web » et « un langage de balisage ».

Grâce au HTML vous allez pouvoir, dans votre navigateur (Firefox, Chrome, Opera,....), afficher du texte, afficher des images, proposer des hyperliens (liens vers d'autres pages web), afficher des formulaires et même maintenant afficher des vidéos (grâce à la dernière version du HTML, l'HTML5).

HTML n'est pas un langage de programmation (comme le Python par exemple), ici, pas question de conditions, de boucles....c'est un langage de description.

Pour aborder le HTML, nous allons, dans un premier temps utiliser le site `jsfiddle.net`.

Après avoir lancé votre navigateur web, tapez `http://jsfiddle.net/` dans la barre d'adresse.

Vous devriez voir apparaître ceci :

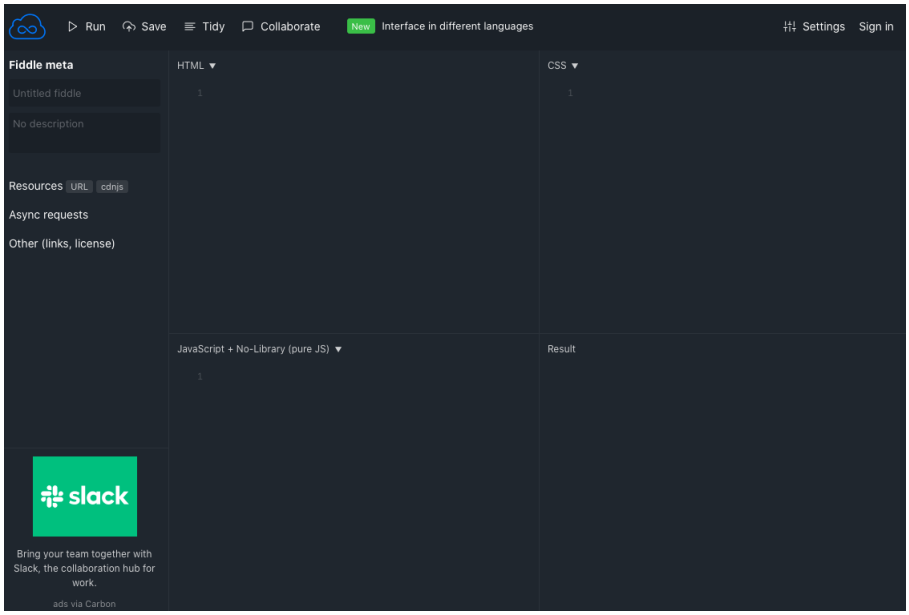


FIGURE 3 – http ://jsfiddle.net/

Nous allons pour l'instant uniquement utiliser la fenêtre « HTML » et la fenêtre « Result ».

Exercice 2 : Écrivez le code HTML suivant :

Premières lignes en HTML

```
1 <h1>Hello World! Ceci est un titre</h1>
2 <p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
```

Qu'est-ce qui s'affiche dans la fenêtre après avoir cliqué sur "Run" ?

Comme déjà évoqué ci-dessus, en HTML tout est une histoire de balise que l'on ouvre et que l'on ferme. Une balise ouvrante est de la forme `<nom_de_la_balise>`, les balises fermantes sont de la forme `</nom_de_la_balise>`.

En observant attentivement le code, vous devriez forcément remarquer que toute balise ouverte doit être refermée à un moment ou un autre. La balise ouvrante et la balise fermante peuvent être sur la même ligne ou pas, cela n'a aucune espèce d'importance, la seule question à se poser ici est : ai-je bien refermé toutes les balises que j'ai ouvertes ?

Enfin pour terminer avec les généralités sur les balises, il est important de savoir qu'une structure du type :

```
<balise1>
<balise2>
</balise1>
</balise2>
```

est interdite, la `balise2` a été ouverte après la balise1, elle devra donc être refermée avant la `balise1`.

En revanche, l'enchaînement suivant est correct :

```
<balise1>
<balise2>
</balise2>
</balise1>
```

Notez que dans notre exemple nous respectons bien cette règle « d'imbrication » des balises avec la balise `<p>` et la balise `<strong>`.

Il est important de comprendre que chaque balise a une signification qu'il faut bien respecter (on parle de la sémantique des balises). Par exemple le texte situé entre la balise ouvrante et fermante `<h1>` est obligatoirement un titre important (il existe des balises `<h2>`, `<h3>`.....qui sont aussi des titres, mais des titres moins importants (sous-titre)). La balise `<p>` permet de définir des paragraphes, enfin, la balise `<strong>` permet de mettre en évidence un élément important.

Vous devez aussi savoir qu'il existe des balises qui sont à la fois ouvrantes et fermantes (`<balise/>`) : un exemple, la balise permettant de sauter une ligne, la balise `<br/>` (balise qu'il faut d'ailleurs éviter d'utiliser par différentes raisons que nous n'aborderons pas ici).

Il est possible d'ajouter des éléments à une balise ouvrante, on parle d'attribut. Une balise peut contenir plusieurs attributs :

`<ma_balise attribut_1= "valeur_1" attribut_2="valeur_2">`

Il existe beaucoup d'attributs différents, nous allons nous contenter de 2 exemples avec l'attribut `id` (`id` pour identifiant) et `class`. Nous verrons l'intérêt de ces attributs dans l'activité suivante.

Exercice 3 : Écrivez le code HTML suivant :

```html
<h1>Ceci est un titre</h1>
<h2 class="titre_1">Ceci est un sous titre</h2>
<p id="para_1">Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
```

Qu'est-ce qui s'affiche dans la fenêtre après avoir cliqué sur "Run" ?

Le HTML n'a pas été conçu pour gérer la mise en page (c'est possible, mais c'est une mauvaise pratique). Le HTML s'occupe uniquement du contenu et de la sémantique, pour tout ce qui concerne la mise en page et l'aspect « décoratif » (on parle du « style » de la page), on utilisera le CSS (Cascading Style Sheets).

Dans JSFIDDLE, il est possible d'écrire du CSS dans la fenêtre en haut à droite.

exercice

Écrivez le code HTML suivant :

```html
<h1>Ceci est un titre</h1>
<h2>Ceci est un sous titre</h2>
<p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
```

Écrivez le code CSS suivant :

```css
h1
{
  text-align: center;
  background-color: red;
}
h2
{
  font-family: Verdana;
  font-style: italic;
  color: green;
}
```

Qu'est-ce qui s'affiche dans la fenêtre après avoir cliqué sur "Run" ?

Dans l'exercice précédent, les propriétés « `text-align` » et « `background-color` » seront appliquées au contenu de toutes les balises de type `h1` (avec respectivement les valeurs « `center` » et « `red` »).....

Exercice 4 : Écrivez le code HTML suivant :

```html
<h1>Ceci est un titre</h1>
<h2>Ceci est un sous titre</h2>
<p id="para_1">Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
```

Écrivez le code CSS suivant :

```css
#para_1
{
  font-style: italic;
  color: green;
}
```

Qu'est-ce qui s'affiche dans la fenêtre ? Que remarquez-vous ?

Il est donc possible de cibler un paragraphe et pas un autre en utilisant l'`id` du paragraphe (en CSS l'`id` se traduisant par le signe `#`).

Il est aussi possible d'utiliser l'attribut `class` à la place de l'`id`. Dans le CSS on utilisera le point `.` à la place du `#`.

La différence entre `"id"` et `"class"` n'est pas très importante.

L'attribut `"class"` permet de donner le même nom à plusieurs reprises dans une même page.

Si nous avions eu un 3e paragraphe, nous aurions pu avoir : `<p class="para_1">Voici un 3e paragraphe</p>`, mais nous n'aurions pas pu avoir : `<p id="para_1"> Voici un 3e paragraphe </p>`, car le nom `para_1` a déjà été utilisé pour le 1er paragraphe.

JSFIDDLE est un très bel outil, mais il ne peut pas être utilisé pour la réalisation d'un vrai site internet (ou d'une vraie application web).

Nous allons créer 2 fichiers : un fichier qui contiendra du HTML (`index.html`) et un fichier qui contiendra du CSS (`style.css`).

Exercice 5 :

À l'aide d'un éditeur de texte, créer un nouveau fichier.

Sauvegardez-le en précisant son nom, par exemple "index.html".

Écrivez le code suivant dans votre éditeur de texte (sans oublier de sauvegarder quand vous avez terminé) :

```
<!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>
```

Testez votre code à l'aide d'un navigateur web (Firefox ou Chrome) en "double-cliquant" sur le fichier index.html

Dans l'exemple du "À faire vous-même 5", vous reconnaissez le code se trouvant entre les balises `<body>` :

```
<body>
......
</body>
```

Tout votre code HTML devra se trouver entre ces 2 balises.

Le reste des balises devraient vous êtes inconnues. Passons-les en revue :

La première ligne (`<!doctype html>`) permet d'indiquer au navigateur que nous utiliserons la dernière version du HTML, le fameux HTML5.

La balise `<html>` est obligatoire, l'attribut `lang="fr"` permet d'indiquer au navigateur que nous utiliserons le français pour écrire notre page.

Les balises `<head>...</head>` délimitent ce que l'on appelle l'en-tête. L'en-tête contient, dans notre exemple, 2 balises : la balise `<meta charset="utf-8">` qui permet de définir l'encodage des caractères (`utf-8`) et la balise `<title>` qui définit le titre de la page (attention ce titre ne s'affiche pas dans le navigateur, ne pas confondre avec la balise `<h1>`).

Exercice 6 :

Toujours à l'aide d'un éditeur de texte, vous allez créer un fichier qui va contenir le CSS de notre page (par exemple style.css). Complétez ce fichier à l'aide du code suivant :

```
h1
{
  text-align: center;
  background-color: red;
}
p
{
  font-family: Verdana;
  font-style: italic;
  color: green;
}
```

Pour l'instant notre CSS ne sera pas appliqué à notre page, pour ce faire, il faut modifier notre code HTML en ajoutant une ligne qui va permettre d'associer notre code CSS à notre page.

Modifiez le code HTML avec la ligne suivante `<link rel="stylesheet" href="style.css">` :

```html
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Voici mon site</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Hello World! Ceci est un titre</h1>
    <p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
  </body>
</html>
```

Testez votre code à l'aide d'un navigateur web en "double-cliquant" sur le fichier index.html

Dans l'exemple que nous venons de voir, les fichiers "index.html" et "style.css" se trouvent dans le même dossier. Il est souvent utile de placer les fichiers CSS dans un dossier "CSS". Il faudra alors modifier le code HTML en conséquence :

```html
<link rel="stylesheet" href="CSS/style.css">
```

Pour terminer, voici quelques balises très utilisées :

La balise a

```html
<a href="mon_autre_page.html">Cliquez ici pour vous rendre sur mon autre page</a>
```

La balise a permet de créer des liens hypertextes, ce sont ces liens hypertextes qui vous permettent de "voyager" entre les pages d'un site ou entre les sites. Les liens hypertextes sont par défaut soulignés et de couleur bleue (modifiable grâce au CSS). La balise a possède un attribut **href** qui a pour valeur le chemin du fichier que l'on cherche à atteindre ou l'adresse du site cible : exemple :

```html
<a href="http://www.google.fr">Cliquez ici pour vous rendre sur google.fr</a>'.
```

Entre la balise ouvrante et fermante, on trouve le texte qui s'affichera à l'écran (c'est ce texte qui est souligné et de couleur bleue).La balise peut sans problème se trouver en plein milieu d'un paragraphe.

1. La balise image

Comme vous devez déjà vous en douter, la balise image sert à insérer des.....images :

```html
<img src="mon_image.jpg" alt="avion"/>
```

La balise **img** est à la fois ouvrante et fermante comme la balise **br**. Elle possède 2 attributs :

- **src** qui doit être égal au nom du fichier image (ou au chemin si le fichier image se trouve dans un autre dossier).
- **alt** qui doit être égal à une description de votre image (cet attribut est utilisé notamment par les systèmes de description des pages web utilisées par les non-voyants), il faut donc systématiquement renseigner cet attribut.

Les balises form, input et button

Les formulaires sont des éléments importants des sites internet, ils permettent à l'utilisateur de transmettre des informations. Un formulaire devra être délimité par une balise form (même si ce n'est pas une obligation) :

```
<form>
....
</form>
```

Il existe différentes balises permettant de construire un formulaire, notamment la balise input. Cette balise possède un attribut type qui lui permet de jouer des rôles très différents.

La balise **button** nous sera aussi d'une grande utilité.

Exercice 7 :

Créez un fichier html contenant le code suivant :

```html
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Voici mon site</title>
  </head>
  <body>
    <form>
      <p>voici un champ de texte : <input type="text"/></p>
      <p>voici une checkbox <input type="checkbox"/></p>
      <button>Cliquez ici !</button>
    </form>
  </body>
</html>
```

Testez votre code à l'aide d'un navigateur web en "double-cliquant" sur le fichier html que vous venez de créer.

Les balises `div` et `span`

Ces 2 balises sont très utilisées (surtout la balise `div`). Pourtant, il faudrait, autant que possible, les éviter, pourquoi ?

Parce qu'elles n'ont aucune signification particulière, ce sont des balises dites "génériques".

À quoi servent-elles alors ?

À organiser la page, à regrouper plusieurs balises dans une même entité.

Pourquoi 2 balises génériques ?

Parce que `div` est une balise de type "block" et que `span` est une balise de type "inline" !

Sans vouloir trop entrer dans les détails, il faut bien comprendre que l'ordre des balises dans le code HTML a une grande importance. Les balises sont affichées les unes après les autres, on parle du flux normal de la page.

C'est ici qu'entrent en jeu les balises de type "block" et les balises de type "inline".

Les contenus des balises de type "block" (`p`, `div`, `h1`,...) se placent sur la page web les uns en dessous des autres. Les contenus des balises de type "inline" (`strong`, `img`, `span`, `a`) se placent sur la page web les uns à côté des autres. Cela revient à dire qu'une balise de type "block" prend toute la largeur de la page alors qu'une balise de type "inline" prend juste la largeur qui lui est nécessaire.

Exercice 8 :

Créez un fichier html contenant le code suivant :

```html
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Voici mon site</title>
  </head>
  <body>
    <div>div est une balise de type "block"</div>
    <p>la balise p est une autre balise de type block</p>
    <span>En revanche, span est une balise de type "inline"</span>
    <a href="www.google.fr">Et voici une autre balise de type "inline"</a>
    <h1>h1 est bien une balise de type "block"</h1>
    <span>la malheureuse balise span est contrainte de se placer en dessous </span>
  </body>
</html>
```

Testez ce code

4 Interaction avec l'utilisateur dans une page web : utilisation du JavaScript

Nous avons déjà pu nous familiariser avec le couple HTML-CSS, en faite, le couple est plutôt un trio, car aujourd'hui un développeur web ne peut pas faire l'impasse sur le JavaScript .

Notre but ici n'est pas d'apprendre un nouveau langage de programmation, mais juste d'étudier quelques exemples d'utilisation du JavaScript, notamment dans le cas des interactions entre un utilisateur et une page web.

Avant d'entrer dans le vif du sujet, un petit rappel historique : JavaScript a été créé en dix jours par Brendan Eich en 1995. Malgré son nom, JavaScript n'a rien à voir avec le langage Java, même si Brendan Eich affirme s'être inspiré

de nombreux langage, dont Java, pour mettre au point JavaScript. Après des débuts chaotiques, il est, comme déjà dit plus haut, devenu incontournable dans le développement web.

Commençons par mettre en place notre environnement de travail :

Exercice 9 :

Dans votre répertoire de travail, créez 3 fichiers : index.html, style.css et script.js

Après avoir ouvert le fichier index.html à l'aide d'un éditeur de texte, saisissez le code ci-dessous :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <p>Voici une page web qui ne fait pas grand-chose</p>
  </body>
  <script src="script.js"></script>
</html>
```

Rien de très nouveau dans ce code, à part le

```
<script src="script.js"></script>
```

qui permet d'exécuter le programme JavaScript contenu dans le fichier "script.js"

Après avoir ouvert le fichier style.css à l'aide d'un éditeur de texte, saisissez le code ci-dessous :

```
h1{
  text-align: center;
}
```

Après avoir ouvert le fichier script.js à l'aide d'un éditeur de texte, saisissez le code ci-dessous :

```
alert("Le JavaScript fonctionne !")
```

Afin d'afficher la page web que nous venons de créer dans un navigateur web, cliquez sur le fichier "index.html"

Comme vous pouvez le constater, la page web s'affiche bien, mais nous avons en plus une fenêtre (souvent qualifiée de surgissante ou pop-up en anglais) qui apparait. L'apparition de cette fenêtre est bien évidemment due à l'instruction "alert" présente dans le JavaScript.

Le but ici n'étant pas d'apprendre à programmer en JavaScript, nous nous contenterons pour le moment de cette simple instruction "alert". Evidemment JavaScript permet de faire bien plus de choses. En effet on retrouve en JavaScript les grands classiques des langages de programmation : variable, condition, boucle, fonction,...Si vous voulez en apprendre plus sur la programmation en JavaScript, je vous invite à consulter le site openclassrooms

Dans l'exemple ci-dessus, l'instruction "alert" est exécutée dès l'ouverture de la page web, il est tout à fait possible de lier l'exécution de certaines instructions JavaScript à l'action d'un utilisateur (par exemple un clic sur un bouton).

Exercice 10 :

Modifiez le code HTML comme suit :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <p>Voici une page web qui ne fait pas grand chose</p>
    <button onclick="maFonction()">Cliquer ici</button>
  </body>
  <script src="script.js"></script>
</html>
```

Modifiez le code JavaScript comme suit :

```
function maFonction() {
  alert("Le JavaScript fonctionne !")
}
```

Testez cette nouvelle page en cliquant sur le fichier "index.html".

Comme vous pouvez le constater, l'instruction "alert" n'est plus exécutée à l'ouverture de la page web, mais uniquement dans le cas où l'utilisateur clique sur le bouton.

On a associé au bouton un événement "onclick", en cas de clic sur la souris, la fonction JavaScript "maFonction()" est exécutée. Si on s'intéresse au code JavaScript, on retrouve bien une fonction "maFonction()" ("function maFonction()..." en JavaScript est équivalent à un "def maFonction() :" en Python). Entre l'accolade ouvrante et l'accolade fermante (qui délimite la fonction), on retrouve uniquement notre instruction "alert". À l'ouverture de la page web cette instruction "alert" n'est pas exécutée, car elle se trouve dans une fonction. Le clic sur le bouton entraine l'exécution de la fonction "maFonction()" et donc de l'instruction "alert".

Il est évidemment possible de faire des choses bien plus complexes que l'affichage d'un simple pop-up avec JavaScript. Il est possible de modifier le style d'une balise, de modifier la classe (CSS) d'une balise ou encore de modifier le contenu d'une balise, voici quelques exemples :

Exercice 11 :

Modifiez le code HTML comme suit :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <p id="monPara">Voici une page web qui ne fait pas grand chose</p>
    <button onclick="maFonction()">Cliquer ici</button>
  </body>
  <script src="script.js"></script>
</html>
```

Modifiez le code JavaScript comme suit :

```
function maFonction() {
  document.querySelector("#monPara").style.color="red";
}
```

Testez cette nouvelle page en cliquant sur le fichier "index.html".

Dans l'exemple ci-dessous, nous avons déjà ajouté un id ("monPara") à la balise "p" dans notre code HTML. Dans le code JavaScript, la ligne

```
document.querySelector("#monPara").style.color="red";
```

permet de modifier le style de la balise ayant pour id "monPara" : la couleur du texte devient rouge. Comme cette modification du style se trouve dans la fonction `maFonction()`, cette modification sera effective uniquement si l'utilisateur appuie sur le bouton.

Il est possible de travailler plus "proprement" en utilisant les classes CSS :

Exercice 12 :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <p id="monPara">Voici une page web qui ne fait pas grand-chose</p>
    <button onclick="foncRouge()">Rouge</button>
    <button onclick="foncVert()">Vert</button>
```

```
  </body>
  <script src="script.js"></script>
</html>
```

Modifiez le code JavaScript comme suit :

```
function foncRouge() {
  document.querySelector("#monPara").classList.remove("vert");
  document.querySelector("#monPara").classList.add("rouge");
}
function foncVert() {
  document.querySelector("#monPara").classList.remove("rouge");
  document.querySelector("#monPara").classList.add("vert");
}
```

Modifiez le code CSS comme suit :

```
h1{
  text-align: center;
}
.rouge {
  color:red;
  font-size:20px;
}
.vert {
  color:green;
  font-size:30px;
}
```

Après avoir analysé le code ci-dessus, testez cette nouvelle page en cliquant sur le fichier index.html

Dans l'exemple ci-dessus, nous avons maintenant 2 boutons, un clic sur le bouton "vert", permet d'exécuter la fonction "foncVert()", un clic sur le bouton "rouge", permet d'exécuter la fonction "foncRouge()", jusque là, rien de vraiment nouveau. La fonction JavaScript "foncVert()" permet de modifier la classe CSS de la balise ayant pour id "monPara". Dans un premier temps, la ligne

```
document.querySelector("#monPara").classList.remove("rouge");
```

permet de supprimer l'association entre la balise d'id "monPara" et la classe CSS "rouge" (si cette association n'existe pas, cette ligne n'a aucun effet). Dans un deuxième temps, on associe la classe CSS "vert" avec la balise d'id "monPara" avec la ligne

```
document.querySelector("#monPara").classList.add("vert");
```

Le principe est identique avec la fonction "foncRouge()".
Il est également possible de modifier le contenu d'une balise HTML :

Exercice 13 :
Modifiez le code HTML comme suit :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <p id="monPara">Voici une page web qui ne fait pas grand chose</p>
    <button onclick="modifMessage()">Cliquez ici</button>
  </body>
  <script src="script.js"></script>
</html>
```

Modifiez le code JavaScript comme suit :

```
function modifMessage() {
  document.querySelector("#monPara").innerHTML = "Bravo, vous avez clique sur le
      bouton !"
}
```

Après avoir analysé le code ci-dessus, testez cette nouvelle page en cliquant sur le fichier index.html.

Le contenu de la balise ayant pour id "monPara" est modifié grâce à la ligne

```
document.querySelector("#monPara").innerHTML = "Bravo, vous avez clique sur le bouton !"
}
```

Il existe d'autres événements que "onclick", par exemple, il est possible de détecter le "survol" par le curseur de la souris d'un élément HTML.

Exercice 14 :

Modifiez le code HTML comme suit :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le trio</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Le trio : HTML, CSS et JavaScript</h1>
    <div onmouseover="foncEntre()" onmouseout="foncQuitte()"  id="maDiv"><p>Survolez-moi</p></div>
  </body>
  <script src="script.js"></script>
</html>
```

Modifiez le code JavaScript comme suit :

```
function foncEntre(){
  document.querySelector("#maDiv").classList.remove("blanc");
  document.querySelector("#maDiv").classList.add("rouge");
}
function foncQuitte() {
  document.querySelector("#maDiv").classList.remove("rouge");
  document.querySelector("#maDiv").classList.add("blanc");
}
```

Modifiez le code CSS comme suit :

```
h1{
  text-align : center;
}
p{
  text-align : center;
}
#maDiv{
  width : 200px;
  height : 100px;
  margin : 0 auto;
  border : 2px solid black;
}
.rouge {
  background-color :red;
}
.blanc {
  background-color : white;
}
```

Après avoir analysé le code ci-dessus, testez cette nouvelle page en cliquant sur le fichier index.html.

"onemouseover" correspond bien au survol par le curseur de la souris d'un élément HTML. L'événement "onemouseout" est lui déclenché quand le curseur de la souris quitte un élément HTML donné.

Il existe beaucoup d'autres événements que nous n'aborderons pas ici. Si vous voulez en savoir plus, vous pouvez consulter ce site .

En résumé, le code HTML permet de générer des éléments graphiques qui seront affichés par un navigateur web, mais pas seulement : il est aussi possible de mettre en place dans le code HTML des événements. Un événement donné pourra déclencher l'exécution d'instructions JavaScript.

5 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` » (en fait c'est plus compliqué, pour les puristes nous diront donc que la communication va être établie avec le serveur `www` du domaine `google.fr`, mais bon, pour la suite nous pourrons nous contenter de l'explication « simplifiée »).

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

client : bonjour `www.google.fr` (ou bonjour `www` se trouvant dans le domaine 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,...).

FIGURE 4 – Un serveur

Afin d'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 ou encore Facebook 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").

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".

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 ».

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.

Différents langages de programmation peuvent être utilisés « côté serveur » afin de permettre au serveur de générer lui-même le code HTML à envoyer. Le plus utilisé encore aujourd'hui se nomme PHP. D'autres langages sont utilisables côté serveur (pour permettre la génération dynamique de code HTML) : Java, Python...

Voici un exemple très simple de code en PHP :

```
1 <?php
2 $heure = date("H:i");
3 echo '<h1>Bienvenue sur mon site</h1>
4       <p>Il est '.$heure.'</p>';
```

FIGURE 5 – Une salle de serveur

```
?>
```

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

6 Protocole 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 (nous verrons ce deuxième cas à la fin de cette activité).
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 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

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 :

```
1  HTTP/1.1 200 OK
2  Date: Thu, 15 feb 2019 12:02:32 GMT
3  Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
4  Connection: close
5  Transfer-Encoding: chunked
6  Content-Type: text/html; charset=ISO-8859-1
7  <!doctype html>
8  <html lang="fr">
9  <head>
10 <meta charset="utf-8">
11 <title>Voici mon site</title>
12 </head>
13 <body>
14  <h1>Hello World! Ceci est un titre</h1>
15 <p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
16 </body>
17 </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 : 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 1re 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 :

```
1  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, n'hésitez pas à faire vos propres recherches). "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, lighttpd...) mais, aux dernières nouvelles, Apache est toujours le plus populaire puisqu'il est installé sur environ la moitié des serveurs web mondiaux !
Le "HTTPS" est la version "sécurisée" du protocole HTTP. Par "sécurisé" en entend que les données sont chiffrées avant d'être transmises sur le réseau.
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 de chiffrement (nous aborderons cette notion de clé de chiffrement l'année prochaine, en terminale). 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 incompréhensible à cause du chiffrement (seuls le client et le serveur sont aptes à déchiffrer ces données)

D'un point vu 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 Formulaire d'une page Web (version Flask)

Comme déjà évoqué dans la partie consacrée au modèle client-serveur, un serveur web (aussi appelé serveur HTTP) permet de répondre à une requête HTTP effectuée par un client (très souvent un navigateur web). Nous allons travailler avec le serveur web qui est installé sur votre ordinateur. Nous allons donc avoir une configuration un peu particulière puisque le client et le serveur vont se trouver sur la même machine. Cette configuration est classique lorsque l'on désire effectuer de simples tests. Nous aurons donc 2 logiciels sur le même ordinateur : le client (navigateur web) et le serveur (serveur web), ces 2 logiciels vont communiquer en utilisant le protocole HTTP. Il existe de nombreux serveurs web, mais le plus utilisé se nomme Apache. Nous n'allons pas utiliser Apache, car nous allons travailler avec le framework Python Flask. Ce framework va nous permettre de générer des pages web côté serveur, il possède son propre serveur web.

Nous allons commencer par un cas très simple où le serveur va renvoyer au client une simple page HTML statique (ne pas hésiter à consulter la partie consacrée au modèle client-serveur pour plus de précision sur ce terme "statique").

Exercice 15 :

Dans votre répertoire personnel, créez un répetoire nommé "flask".

À l'aide de Spyder, créez un fichier Python "`views.py`" (ce fichier devra être sauvegardé dans le répertoire "flask" précédemment créé). Saisissez le code suivant dans le fichier "`views.py`"

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def index():
7   return "<p>Tout fonctionne parfaitement</p>"
8
9 app.run(debug=True)
```

Après avoir exécuté le programme ci-dessus, ouvrez votre navigateur web et tapez dans la barre d'adresse "`localhost:5000`".

Vous devriez voir la phrase "`Tout fonctionne parfaitement`" s'afficher dans votre navigateur.

Une petite explication s'impose à propos du "`localhost:5000`" : comme nous l'avons déjà dit, notre serveur et notre client se trouvent sur la même machine, avec le "`localhost`", on indique au navigateur que le serveur web se trouve sur le même ordinateur que lui (on parle de machine locale). Dans un cas normal, la barre d'adresse devrait être renseignée avec l'adresse du serveur web. Le "`5000`" indique le port, nous n'étudierons pas cet aspect des choses ici, vous devez juste savoir que le "`5000`" doit suivre le "`localhost`".

Stoppez l'exécution du programme dans Spyder.

Essayons de comprendre en détail ce qui s'est passé :

En exécutant le programme Python ci-dessus, le framework Flask a lancé un serveur web. Ce serveur web attend des requêtes HTTP sur le port 5000. En ouvrant un navigateur web et en tapant "`localhost:5000`", nous faisons une requête HTTP, le serveur web fourni avec Flask répond à cette requête HTTP en envoyant une page web contenant uniquement "`<p>Tout fonctionne parfaitement</p>`".

Reprenons le programme Python ligne par ligne :

```
from flask import Flask
```

Nous importons la bibliothèque Flask

```
app = Flask(__name__)
```

Nous créons un objet `app` : cette ligne est systématique nécessaire.

```
@app.route('/')
```

Nous utilisons ici un décorateur (cette notion de décorateur ne sera pas traitée en NSI). Vous devez juste comprendre la fonction qui suit ce décorateur ("index"), sera exécutée dans le cas où le serveur web recevra une requête HTTP avec une URL correspondant à la racine du site ('/'), c'est à dire, dans notre exemple, le cas où on saisie dans la barre d'adresse "`localhost:5000/`" (ou simplement "`localhost:5000`") Nous verrons ci-dessous un exemple avec une URL autre que '/'.

```
def index():
    return "<p>Tout␣fonctionne␣parfaitement</p>"
```

En cas de requête HTTP d'un client avec l'URL "`/`", le serveur renvoie vers le client une page HTML contenant uniquement la ligne "`<p>Tout fonctionne parfaitement</p>`".

```
app.run(debug=True)
```

Cette ligne permet de lancer le serveur, elle sera systématiquement présente.

Exercice 16 :

À l'aide de Spyder, modifiez le fichier Python "`views.py`" :

```
1  from flask import Flask
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7    return "<p>Tout fonctionne parfaitement</p>"
8  @app.route('/about')
9  def about():
10   return "<p>Une autre page</p>"
11 app.run(debug=True)
```

Après avoir exécuté le programme ci-dessus, saisissez "`localhost:5000/about`" dans la barre d'adresse de votre navigateur.

Comme vous pouvez le constater, le serveur nous renvoie dans ce cas une autre page. Évidemment l'URL racine ("`/`") reste disponible, vous pouvez passer d'une page à l'autre en modifiant l'URL dans la barre d'adresse ("`localhost:5000`" ou "`localhost:5000/about`")

Écrire le code HTML qui devra être renvoyé au client dans le programme Python n'est pas très pratique, Flask propose une autre solution bien plus satisfaisante :

Exercice 17 :

Dans votre répertoire "Flask", créez un répertoire "templates". Dans ce répertoire templates, créez un fichier `index.html`. Saisissez le code HTML ci-dessous dans ce fichier index.html

```
1  <!doctype html>
2  <html lang="fr">
3    <head>
4      <meta charset="utf-8">
5      <title>Ma page</title>
6    </head>
7    <body>
8      <h1>Mon super site</h1>
9      <p>Tout fonctionne parfaitement</p>
10   </body>
11 </html>
```

et modifiez le programme `views.py` comme suit :

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def index():
7      return render_template("index.html")
8
9  app.run(debug=True)
```

Relancez le programme Python et tapez "localhost:5000" dans la barre d'adresse de votre navigateur

Le serveur renvoie maintenant au client la page HTML correspondant au fichier "index.html" qui a été créé dans le répertoire "templates". Attention, les fichiers HTML devront systématiquement se trouver dans un répertoire nommé "templates".

N. B. le "debug=True" de la dernière ligne permet de modifier les fichiers HTML sans être obligé de redémarrer le programme "views.py".

Pour l'instant notre site est statique : la page reste identique, quelles que soient les actions des visiteurs. Flask permet de créer des pages dynamiques :

- le client (le navigateur web) envoie une requête HTTP vers un serveur web
- en fonction de la requête reçue et de différents paramètres, Flask "fabrique" une page HTML différente
- le serveur web associé à Flask envoie la page nouvellement créée au client
- une fois reçue, la page HTML est affichée dans le navigateur web

Exercice 18 :

Modifiez le programme views.py comme suit :

```
1  from flask import Flask, render_template
2  import datetime
3
4  app = Flask(__name__)
5
6  @app.route('/')
7  def index():
8    date = datetime.datetime.now()
9    h = date.hour
10   m = date.minute
11   s = date.second
12   return render_template("index.html", heure = h, minute = m, seconde = s)
13
14 app.run(debug=True)
```

Dans le programme ci-dessous nous importons le module "datetime" afin de pouvoir déterminer la date et l'heure courante. Le

```
date = datetime.datetime.now()
```

nous permet de récupérer la date et l'heure courante

```
h = date.hour
m = date.minute
s = date.second
```

Après l'exécution des 3 lignes ci-dessus, les variables h, m et s contiennent l'heure courante.

La fonction "render_template"

```
return render_template("index.html", heure = h, minute = m, seconde = s)
```

contient 3 paramètres de plus par rapport à l'exemple du "À faire vous-même 6" : le paramètre "heure", le paramètre "minute" et le paramètre "seconde", nous allons retrouver ces 3 paramètres dans le fichier HTML.

Exercice 19 :

Modifiez le fichier "index.html" comme suit :

```
1  <!doctype html>
2  <html lang="fr">
3    <head>
4      <meta charset="utf-8">
5      <title>Utilisation de Flask</title>
6    </head>
7    <body>
8      <h1>Mon super site</h1>
9      <p>Le serveur fonctionne parfaitement, il est {{heure}} h {{minute}} minutes et {{
           seconde}} secondes</p>
10   </body>
11 </html>
```

Testez ces modifications en saisissant "localhost:5000" dans la barre de votre navigateur web.

Nous avons bien une page dynamique, puisqu'à chaque fois que vous actualisez la page dans votre navigateur, l'heure courante s'affiche : à chaque fois que vous actualisez la page, vous effectuez une nouvelle requête et en réponse à cette requête, le serveur envoie une nouvelle page HTML.

Le fichier "index.html" ne contient donc pas du HTML (même si cela ressemble beaucoup à du HTML), car les paramètres {{heure}}, {{minute}} et {{seconde}} n'existent pas en HTML. Le fichier "index.html" contient en fait un langage de template nommé Jinja. Jinja ressemble beaucoup au HTML, mais il rajoute beaucoup de fonctionnalités par rapport au HTML (notamment les paramètres entourés d'une double accolade comme {{heure}}). Si vous utilisez Jinja seul (sans un framework comme Flask), les paramètres ne seront pas remplacés et votre navigateur affichera "Le serveur fonctionne parfaitement, il est {{heure}} h {{minute}} minutes et {{seconde}} secondes".

Nous allons maintenant nous intéresser à la gestion des formulaires.

Exercice 20 :

Modifiez le fichier "index.html" comme suit :

```
<!doctype html>
  <html lang="fr">
    <head>
      <meta charset="utf-8">
      <title>Le formulaire</title>
    </head>
    <body>
      <form action="http://localhost:5000/resultat" method="post">
          <label>Nom</label> : <input type="text" name="nom" />
          <label>Prenom</label> : <input type="text" name="prenom" />
          <input type="submit" value="Envoyer" />
      </form>
    </body>
  </html>
```

et créez un fichier "resultat.html" (dans le répertoire "templates"), ce fichier devra contenir le code suivant :

```
<!doctype html>
  <html lang="fr">
    <head>
      <meta charset="utf-8">
      <title>Resultat</title>
    </head>
    <body>
      <p>Bonjour {{prenom}} {{nom}}, j'espere que vous allez bien.</p>
    </body>
  </html>
```

Modifiez le fichier views.py comme suit :

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
  return render_template('index.html')

@app.route('/resultat',methods = ['POST'])
def resultat():
  result = request.form
  n = result['nom']
  p = result['prenom']
  return render_template("resultat.html", nom=n, prenom=p)

app.run(debug=True)
```

Après avoir relancé "views.py", testez cet exemple en saisissant "localhost:5000" dans la barre d'adresse de votre navigateur web.

Si vous saisissez, par exemple, "Turing" et "Alan" dans les champs "Nom" et "Prénom" du formulaire, vous devriez obtenir le résultat suivant après avoir appuyé sur le bouton "Envoyer" :

Reprenons un par un les événements qui nous ont amenés à ce résultat :

Nous effectuons une requête HTTP avec l'URL "/", le serveur génère une page web à partir du fichier "index.html", cette page, qui contient un formulaire (balise <form action="http://localhost:5000/ resultat" method="post">)

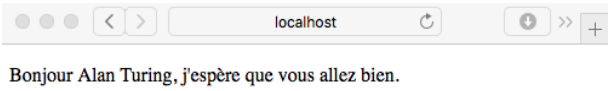


FIGURE 6 – Résultat d'affichage

est envoyée vers le client. On remarque 2 attributs dans cette balise form : `action="http://localhost:5000/resultat"` et `method="post"`. Ces 2 attributs indiquent que le client devra effectuer une requête de type `POST` (la méthode `POST` a déjà été vue dans la partie consacrée au protocole HTTP) dès que l'utilisateur appuiera sur le bouton "Envoyer". Cette requête POST sera envoyée à l'URL `"http://localhost:5000/resultat"` (voir l'attribut `"action"`). Les données saisies dans le formulaire seront envoyées au serveur par l'intermédiaire de cette requête.

N.B Vous avez sans doute remarqué que la méthode à employer pour effectuer la requête HTTP n'est pas précisée dans le `"app.route('/')"`. Si rien n'est précisé, par défaut, c'est la méthode `GET` qui est utilisée.

Intéressons-nous à la fonction `"resultat"`, puisque c'est cette fonction qui sera exécutée côté serveur pour traiter la requête `POST` :

```
def resultat():
 result = request.form
 n = result['nom']
 p = result['prenom']
 return render_template("resultat.html", nom=n, prenom=p)
```

"request.form" est un dictionnaire Python qui a pour clés les attributs "name" des balises "input" du formulaire (dans notre cas les clés sont donc "nom" et "prenom") et comme valeurs ce qui a été saisi par l'utilisateur. Si l'utilisateur saisit "Martin" et "Sophie", le dictionnaire "request.form" sera :

```
{'nom':'Martin', 'prenom':'Sophie'}
```

Le reste du code ne devrait pas vous poser de problème.

Le template "resultat.html" utilise des paramètres "nom" et "prenom".

En réponse à la requête `POST`, le serveur renvoie une page HTML créée à partir du template `"resultat.html"` et des paramètres "nom" et "prenom". Si l'utilisateur a saisi "Turing" et "Alan", le navigateur affichera "Bonjour Alan Turing, j'espère que vous allez bien."

Pour gérer le formulaire, il est possible d'utiliser une méthode HTTP `"GET"` à la place de la méthode `"POST"` :

Exercice 21 :

Modifiez les fichiers comme suit : `index.html`

```
<!doctype html>
 <html lang="fr">
   <head>
     <meta charset="utf-8">
     <title>Le formulaire</title>
   </head>
   <body>
     <form action="http://localhost:5000/resultat" method="get">
         <label>Nom</label> : <input type="text" name="nom" />
         <label>Prenom</label> : <input type="text" name="prenom" />
         <input type="submit" value="Envoyer" />
     </form>
   </body>
 </html>
```

`resultat.html` (le fichier est inchangé)

```
<!doctype html>
 <html lang="fr">
   <head>
     <meta charset="utf-8">
     <title>Resultat</title>
   </head>
   <body>
     <p>Bonjour {{prenom}} {{nom}}, j'espere que vous allez bien.</p>
   </body>
 </html>
```

`views.py`

```python
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route('/')
def index():
 return render_template('index.html')

@app.route('/resultat',methods = ['GET'])
def resultat():
  result=request.args
  n = result['nom']
  p = result['prenom']
  return render_template("resultat.html", nom=n, prenom=p)

app.run(debug=True)
```

Dans "`index.html`", la méthode `POST` a été remplacée par la méthode `GET`. Dans le fichier "`views.py`" nous avons aussi remplacé `POST` par `GET`, et on utilise "`request.args`" à la place de "`request.form`".

Exercice 22 :

Relancez l'exécution de "views.py" et saisissez "`localhost:5000`" dans la barre d'adresse d'un navigateur web. Une fois la page web affichée dans votre navigateur, Saisissez "Alan" pour le prénom et "Turing" pour le nom puis validez en cliquant sur le bouton "Envoyer". Une fois que vous avez cliqué sur le bouton "Envoyer", observez attentivement la barre d'adresse de votre navigateur. Vous devriez obtenir quelque chose qui ressemble à celà :

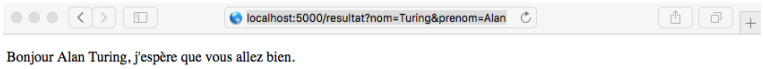


FIGURE 7 – Résultat d'affichage

Vous avez dû remarquer que cette fois-ci, les informations du formulaire sont transmises au serveur par l'intermédiaire de l'URL : `localhost:5000/resultat?nom=Martin&prenom=Sophie`

Dans le cas de l'utilisation d'une méthode "POST" les données issues d'un formulaire sont envoyées au serveur sans être directement visibles, alors que dans le cas de l'utilisation d'une méthode "`GET`", les données sont visibles (et accessibles) puisqu'elles sont envoyées par l'intermédiaire de l'URL.

Les données envoyées par l'intermédiaire d'une méthode "`GET`" peuvent être modifiées directement dans l'URL :

Exercice 23 :

Ouvrez votre navigateur Web et tapez dans la barre d'adresse "`localhost:5000`". Une fois la page web affichée dans votre navigateur, Saisissez "Alan" pour le prénom et "Turing" pour le nom puis validez en cliquant sur le bouton "Envoyer". Une fois que le message "Bonjour Alan Turing, j'espère que vous allez bien." apparait, modifier l'URL : passez de "`localhost:5000/resultat?nom=Turing&prenom=Alan`" à "`localhost:5000/resultat?nom=Berry&prenom=Gérard`", validez votre modification en appuyant sur la touche "Entrée".

Comme vous pouvez le constater, la page a bien été modifiée : "Bonjour Gérard Berry, j'espère que vous allez bien."

Même si dans notre cas cette opération de modification d'URL est inoffensive, vous devez bien vous douter que dans des situations plus complexes, une telle modification pourrait entrainer des conséquences plus problématiques (piratage). Il faut donc éviter d'utiliser la méthode "`GET`" pour transmettre les données issues d'un formulaire vers un serveur.

8 Formulaire d'une page Web (version PHP)

Comme déjà évoqué dans la partie consacrée au modèle client-serveur, un serveur Web (aussi appelé serveur HTTP) permet de répondre à une requête HTTP effectuée par un client (très souvent un navigateur Web). Nous allons travailler avec le serveur Web qui est installé sur votre ordinateur. Nous allons donc avoir une configuration un peu particulière puisque le client et le serveur vont se trouver sur la même machine. Cette configuration est classique lorsque l'on désire effectuer de simples tests. Nous aurons donc 2 logiciels sur le même ordinateur : le client (navigateur Web) et le serveur (serveur Web), ces 2 logiciels vont communiquer en utilisant le protocole HTTP. Le serveur Web que nous allons utiliser se nomme "Apache", c'est un des serveur Web le plus utilisé au monde. Apache a déjà été installé et configuré sur votre ordinateur.

Nous allons commencer par un cas très simple où le serveur va renvoyer au client une simple page HTML statique (ne pas hésiter à consulter la partie consacrée au modèle client-serveur pour plus de précision sur ce terme "statique"). Le serveur Web Apache a été configuré pour qu'il envoie vers le client une page HTML située dans un répertoire nommé "www", ce répertoire "www" devant se trouver dans votre répertoire personnel.

Exercice 24 :

Créez dans votre répertoire personnel un répertoire nommé "www".
Créez un fichier "index.html", placez ce fichier "index.html" dans le répertoire "www" tout nouvellement créé.
Copiez le code HTML ci-dessous dans le fichier "index.html" que vous venez de créer.

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Utilisation d'Apache</title>
  </head>
  <body>
    <p>Le serveur Apache fonctionne parfaitement</p>
  </body>
</html>
```

Ouvrez votre navigateur Web et tapez dans la barre d'adresse "localhost".
Vous devriez voir votre page Web s'afficher.
Une petite explication s'impose à propos du "localhost" : comme nous l'avons déjà dit, notre serveur et notre client se trouvent sur la même machine, avec le "localhost", on indique au navigateur que le serveur Web se trouve sur le même ordinateur que lui (on parle de machine locale). Dans un cas normal, la barre d'adresse devrait être renseignée avec l'adresse du serveur Web.
Pour l'instant, comme déjà dit plus haut, notre site est statique : la page reste identique, quelles que soient les actions des visiteurs. Pour avoir un site dynamique, nous allons exécuter, côté serveur, un programme qui va créer de toute pièce une page HTML, cette page HTML sera ensuite envoyée au client par l'intermédiaire du serveur Web. Il existe différents langages de programmation qui permettent de générer des pages HTML à la volée : Python, Java, Ruby... Dans notre cas, nous allons utiliser le PHP (PHP est un acronyme récursif puisqu'il signifie : PHP Hypertext Preprocessor). Le PHP est un langage très utilisé même si dans le monde professionnel Java, Python, Ruby,... sont préférés au PHP. Il est très important de bien comprendre les processus mis en oeuvre :

- le client (le navigateur Web) envoie une requête HTTP vers un serveur Web
- en fonction de la requête reçue le serveur "fabrique" une page HTML grâce à l'exécution d'un programme écrit en PHP (ou en Python, Java...)
- le serveur Web envoie la page nouvellement créée au client
- une fois reçue, la page HTML est affichée dans le navigateur Web

Le langage PHP est déjà installé sur l'ordinateur que vous utilisez actuellement, nous allons donc pouvoir écrire notre premier programme en PHP. Notez qu'il n'est pas question ici d'apprendre à programmer en PHP, mais juste d'avoir un premier contact avec ce langage (et avec cette notion de Web dynamique)

Exercice 25 :

Après avoir supprimé le fichier "index.html" préalablement créé dans le répertoire "www", créez un fichier "index.php", toujours dans le répertoire "www".
Copiez le code PHP ci-dessous dans le fichier "index.php" que vous venez de créer.

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

Ouvrez votre navigateur Web et tapez dans la barre d'adresse "`localhost`".

Vous devriez avoir une page HTML qui vous donne l'heure, si vous actualisez la page, vous pourrez remarquer que bien évidemment l'heure évolue. Nous avons donc bien une page dynamique : le serveur PHP crée la page Web au moment où elle est demandée. À chaque fois que vous actualisez la page, la page HTML est générée de nouveau.
Vous aurez sans doute remarqué que l'extension "`.html`" a été remplacée par "`.php`". Au moment de la requête, le programme contenu dans ce fichier a été exécuté et la page HTML a été générée. Dans les 2 cas, le fichier se nomme

"index", pourquoi ? Par défaut, le serveur prend en compte un fichier nommé "`index`" ("`index.php`" ou "`index.html`" selon les cas). Si vous voulez nommer votre fichier autrement, il faudra modifier ce que vous avez saisi dans la barre d'adresse de votre navigateur : si par exemple vous voulez nommer votre fichier "`toto.html`" (ou "toto.php"), il faudra saisir dans la barre d'adresse du navigateur "`localhost/toto.html`" (ou "`localhost/toto.php`").

Quelques remarques sur le programme PHP ci-dessus :

"`$heure = date("H:i");`", "`$heure`" est une variable (en PHP les variables commencent par un "$"), cette variable "contient" une chaîne de caractères qui correspond à l'heure courante l'instruction "echo" permet d'afficher la chaîne de caractères qui suit l'instruction. Dans notre cas, la chaîne de caractères est "`<h1>Bienvenue sur mon site</h1> <p> I`" ce qui correspond à du HTML à l'exception de "`$heure`" qui permet d'afficher le contenu de la variable "`$heure`". La page Web générée contiendra le code HTML et le contenu de la variable "`$heure`". Le point "." est, en PHP, l'opérateur de concaténation (alors que par exemple en Python, l'opérateur de concaténation est le "+") Si un client effectue une requête à 18h23, le serveur enverra au client le code HTML ci-dessous :

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

Nous allons maintenant nous intéresser à la gestion des formulaires.

Exercice 26 :

Après avoir supprimé tout le contenu de votre dossier "www", créez dans ce même dossier un fichier "index.html" et un fichier "`trait_form.php`"

Copiez le code HTML ci-dessous dans le fichier "`index.html`" que vous venez de créer :

```
<!doctype html>
<html lang="fr">
  <head>
    <meta charset="utf-8">
    <title>Le formulaire</title>
  </head>
  <body>
    <form action="trait_form.php" method="post">
        <label>Nom</label> : <input type="text" name="nom" />
        <label>Prenom</label> : <input type="text" name="prenom" />
        <input type="submit" value="Envoyer" />
    </form>
  </body>
</html>
```

Copiez le code PHP ci-dessous dans le fichier "`trait_form.php`" que vous venez de créer :

```
<?php
    $n=$_POST['nom'];
    $p=$_POST['prenom'];
    echo "<p>Bonjour ".$p." ".$n.", j'espere que vous allez bien.</p>";
?>
```

Ouvrez votre navigateur Web et tapez dans la barre d'adresse "localhost". Une fois la page Web affichée dans votre navigateur, remplissez le formulaire proposé et validez en cliquant sur le bouton "Envoyer"